AFRL-IF-RS-TR-2004-6
Final Technical Report
January 2004

THE MONITORING, DETECTION, ISOLATION & ASSESSMENT OF INFORMATION WARFARE ATTACKS THROUGH MULTI-LEVEL, MULTI-SCALE SYSTEM MODELING AND MODEL BASED TECHNOLOGY

Arizona State University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. H502

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** JANUARY 2004	**3. REPORT TYPE AND DATES COVERED** Final May 99 – May 01

4. TITLE AND SUBTITLE THE MONITORING, DETECTION, ISOLATION & ASSESSMENT OF INFORMATION WARFARE ATTACKS THROUGH MULTI-LEVEL, MULTI-SCALE SYSTEM MODELING AND MODEL BASED TECHNOLOGY	**5. FUNDING NUMBERS** C - F30602-99-1-0506 PE - 63760E & 62301E PR - H502 TA - 34 WU - 01
6. AUTHOR(S) Nong Ye	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Arizona State University Box 875906 1711 South Rural Road Goldwater Center Room 502 Tempe Arizona 85287	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency, 3701 North Fairfax Drive, Arlington Virginia 22203-1714 AFRL/IFGB, 525 Brooks Road, Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2004-6

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: John C. Faust/IFGB/(315) 330-4544/ John.Faust@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

With the goal of protecting computer and networked systems from various attacks, the following intrusion detection techniques were developed and tested using the 1998 and 2000 MIT Lincoln Lab Evaluation Data: Exponentially Weighted Moving Average techniques for autocorrelated and uncorrelated data to detect anomalous changes in the audit event intensity; a learning and inference algorithm based on a first-order Markov chain model of a normal profile for anomaly detection; two multivariate statistical process control techniques based on chi-square and Canberra distance metrics for anomaly intrusion detection; the technique of probabilistic networks with undirected links to represent the symmetric relations of audit event types during normal activities, build a long-term profile of normal activities, and then perform anomaly detection; and Decision tree techniques to automatically learn intrusion signatures, and to classify information system activities into normal or intrusive for producing useful intrusion warning information. Finally, this report presents a research prototype of an Intrusion Detection System (IDS) integrating the intrusion detection techniques and a process model of a computer and network system.

14. SUBJECT TERMS Intrusion Detection, Data Mining, Statistical Process Control, Anomaly Detection, Computer Security	**15. NUMBER OF PAGES** 150
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

INTRODUCTION

As we increasingly rely on information infrastructures to support critical operations in defense, banking, telecommunication, transportation, electric power and many other systems, intrusions into information systems have become a significant threat to our society with potentially severe consequences [1]. An intrusion compromises the security (e.g., availability, integrity, and confidentiality) of an information system through various means, including denial-of-service, remote-to-local, user-to-root, information probing, and so on [2]. Denial-of-service intrusions make a host or network service unavailable by overloading or disrupting the service. Remote-to-local intrusions gain unauthorized access to a host machine without a legitimate user account on the host machine. User-to-root intrusions happen when a regular user on a host machine obtains privileges normally reserved for a root or super user. Information probing intrusions use programs to scan a network of computers for gathering information or searching for known vulnerabilities.

Layers of defense can be set up against intrusions through prevention, detection, and so on. Firewalls, authentication, and cryptography are some of the on-line intrusion prevention mechanisms that protect information systems from external intrusions [3]. Off-line intrusion prevention efforts focus on methodologies of secure software design and engineering. The on-line prevention mechanisms form a fence around information systems to raise the difficulty level of breaking into information systems. However, the fence can only be raised to such a level that services from information systems are not degraded. Although secure software methodologies will continue to improve, bugs and vulnerabilities in information systems are inevitable due to difficulty in managing the complexity of large-scale information systems during their specification, design, implementation, and installation. Intruders explore bugs and vulnerabilities in information systems to attack information systems. Hence, we expect that some intrusions will be leaked through the fence of prevention and results in intrusions into information systems.

Intrusion detection techniques capture intrusions while they are acting on an information system. Existing intrusion detection techniques fall into two major categories: signature recognition and anomaly detection [4-21]. Signature recognition techniques [6-10], also referred to as misuse detection in some literature [4-5], match activities in an information system with signatures of known intrusions, and signal intrusions when there is a match. For a subject (user, file, privileged program, host, network, etc.) of interest, anomaly detection techniques establish a profile of the subject's normal behavior (norm profile), compare the observed behavior of the subject with its norm profile, and signal intrusions when the subject's observed behavior deviates significantly from its norm profile [11-21]. Therefore, anomaly detection techniques rely on a norm profile, and consider a deviation of the subject's behavior from its norm profile as a symptom of an intrusion. A justification of using anomaly detection for intrusion detection is provided in [15]. A detailed review of the existing intrusion detection techniques can be found in [4-5, 11].

Many intrusions into information systems are manifest through the significantly increased or decreased intensity of events occurring in information systems. For example, in typical denial-of-service attacks, an overwhelming number of service requests may be sent to a server (e.g., web server) of an information system over a short period of time to deplete the computational resource in the server and thus deny the server's ability to respond to users' service requests. Such denial-of-service attacks increase the intensity of events on the server. In many virus or worm attacks through email servers, the number of emails received over a short period of time also increases abruptly during the attacks. Intruders into an information system who have gained super-user privileges can disable many resources in the information system, resulting in the abruptly decreased intensity of events in the information system. Therefore, the early detection of significant changes in the event intensity can help stop many intrusions early on to protect information systems and assure the reliability and QoS of information systems. Chapter 1 presents our work on applying one kind of the SPC techniques – EWMA – to intrusion detection for monitoring and detecting intrusions that are manifest through anomalous changes in the intensity of events in an information system.

Ye, Li, and Emran [22] show that attack detection using a number of consecutive events during a given period produces better performance than using a single event at a given time. Since many cyber attacks require a series of related events to accomplish, it is possible to improve attack detection performance by incorporating the ordering of events. A comparative study on a number of anomaly detection techniques indicates that a technique based on a first-order Markov chain model of event transitions as a norm profile produces comparable performance with a high-order Markov model and the other techniques. However, this technique based on the first-order Markov chain model of a norm profile is still computationally intensive due to its use of the Bayes' parameter estimation for learning the norm profile and a likelihood ratio test for inferring anomalies. Considering large amounts and high frequency of events in a computer and network system, this technique is not applicable to cyber attack detection in real time. Chapter 2 presents an anomaly detection technique based on the first-order Markov chain model of a norm profile with a learning algorithm and an inference algorithm at a low computational cost.

In our previous work [23], we present an anomaly detection technique based on multivariate quality control using a chi-square distance metric to monitor computer audit data of activities in an information system and detect intrusive activities. This technique is developed to overcome: 1) the drawbacks of other statistical-based anomaly detection techniques for intrusion detection, e.g., the sensitivity to the normality assumption and the limitation to univariate analysis, and 2) the scalability problem of conventional multivariate quality control techniques such as Hotelling's T^2 test when it is applied to large amounts of computer audit data for intrusion detection. Our previous work demonstrates the feasibility and promising performance of our technique based on chi-square distance metric for intrusion detection on one set of computer audit data including both normal and intrusive activities. Chapter 3 reports our further work on the robustness of this multivariate quality control technique based on the chi-square distance metric as well as the Canberra distance metric to noises that are typically present in computer audit data.

We develop a probabilistic network technique in chapter 4 to capture and represent the profile of the normal behavior. Probabilistic inference is used for detecting anomalies. Since no specific forms of statistical distributions and distribution-based statistical inferences are used in the profile representation and anomaly detection, our technique overcomes the problem with the normality assumption.

Chapter 5 presents our work on applying the decision tree technique for automatic signature recognition, addressing the feature selection method, noises in data, etc. We briefly review the decision tree technique. Then we define a problem of computer intrusion detection. We also describe different feature selection methods that lead to different designs of the decision tree classifiers, as well as different levels of noises in data. We analyze and discuss the performance results of these different decision tree classifiers for the same training and testing data.

An Intrusion Detection System (IDS) aims in detecting intrusive activities, and gives warnings to the System Security Administrator (SSA). Existing IDSs can be categorized into three types: host-based IDS, network-based IDS and router-based IDS. We will describe these IDSs in chapter 6. Host-based IDSs are usually deployed on individual host-machines to monitor activities on the host machines. Its main advantage is that it can detect intrusions targeting the host machines from both insiders and outsiders. Network-based IDSs are installed in some strategic computers in the network to monitor data packages sent between host machines. Network-based IDSs can detect violations of network security policy, but may not be able to process information such as not only the header portion but also the data portion of data packages so as to reveal specific intrusive activities for accurate detection. Router-based IDSs are installed on routers to monitor data packages passing through routers, thus trying to prevent intrusive data packages from entering the network inside the router. Router-based IDSs are similar to network-based IDSs, and thereby suffer from similar problems.

Chapter 1: EWMA Techniques for Detecting Anomalous Changes in Event Intensity

1-1 EWMA Techniques

SPC techniques have typically been used for monitoring and controlling the quality of manufacturing process. SPC techniques can be univariate or multivariate, and detect changes in the process mean (mean shifts), the process variance (variance changes), the relationships among multiple variables (counter-relationships), and so on [24-25]. We are interested in detecting significant changes of the event intensity for intrusion detection. The event intensity is a single variable measuring one characteristic of events in an information system. Hence, we consider only univariate SPC techniques to detect mean shifts in the event intensity as anomalies or possible intrusions.

Shewhart control charts, CUSUM control charts, and EWMA control charts are univariate SPC techniques that are typically used to detect mean shifts [24-30]. EWMV control charts [25, 31] are designed to detect variance changes, but can also be sensitive to mean shifts. EWMA control charts are robust to the non-normality of data [25, 27]. Since we cannot guarantee the normality of the intensity of events occurring in an information system, we consider only EWMA control charts in this study.

EWMA was first suggested in [27]. More recent descriptions of EWMA can be found in [25-26]. A description of the multivariate EWMA can be found in [32]. If the data contain a sequence of uncorrelated process observations, *x(i)*, the EWMA control chart plots *z(i)* which is computed as follows [25]:

$$z(i) = \lambda x(i) + (1-\lambda) z(i-1) \qquad (1)$$

where $0 < \lambda \leq 1$, *z(i)* is the EWMA statistic for the ith observation, *x(i)*. The mean (μ_z) and variance (σ_z^2) of *z(i)* are [25]:

$$\mu_z = \mu_x \tag{2}$$

$$\sigma_z^2 = \sigma_x^2 \left(\frac{\lambda}{2-\lambda} \right) \tag{3}$$

where μ_x and σ_x are the mean and standard deviation of *x(i)* which can be estimated from training data before testing. The control limits for the EWMA control chart are [25]:

$$\begin{aligned} UCL_z &= \mu_z + L\sigma_z \\ LCL_z &= \mu_z - L\sigma_z \end{aligned} . \tag{4}$$

For the 5% significance level, the L value is 1.96. If *z(i)* falls outside the control limits, an anomaly is detected, and an alarm signal is triggered.

If the data contain a sequence of autocorrelated process observations, *x(i)*, the EWMA statistic in formula (1) can be used to provide a one-step–ahead prediction model of autocorrelated data when the process mean does not drift too quickly [28]. The one-step-ahead prediction for *x(i)* is given by *z(i-1)*. The one-step-ahead prediction error is given by [28]:

$$e(i) = x(i) - z(i-1) \tag{5}$$

The parameter λ in formula (1) can be set by minimizing the sum of squared one-step-ahead prediction errors, *e(i)*, on the training data [28], or in a different way. The sequence of one-step-ahead prediction errors, *e(i)*, are independently distributed with mean zero and the standard deviation σ_e. The EWMA control chart plots *e(i)*. The control limits are given by [28]:

$$\begin{aligned} UCL_e &= L\sigma_e \\ LCL_e &= -L\sigma_e \end{aligned} . \tag{6}$$

The EWMA control chart on *e(i)* is equivalent to the EWMA control chart on *x(i)* with the following control limits [28]:

$$UCL_x(i) = z(i-1) + L\sigma_e(i-1)$$
$$UCL_x(i) = z(i-1) - L\sigma_e(i-1) \tag{7}$$

where L is 1.96 for the 5% significance level, and σ_e can be estimated by calculating a smoothed variance as follows [28]:

$$\sigma_e^2(i) = \alpha e(i)^2 + (1-\alpha)\sigma_e^2(i-1) \tag{8}$$

where $0 < \alpha \leq 1$. It is suggested in [40] that smaller values of α are preferred. Other variations of EWMA for autocorrelated data can also be found in [28].

Events in an information system are usually autocorrelated, because users usually carry out a series of related commands in order to complete a given task. Hence, the data of the event intensity in an information system may be autocorrelated. In this study, both EWMA for autocorrelated data and EWMA for uncorrelated data are applied to the data of the event intensity in an information system. The two EWMA techniques are compared with regard to their performance for intrusion detection.

1-2 Application

This section describes the intrusion detection problem, including the data source, the observation of the event intensity, the training data, the testing data, and the application of the EWMA techniques to the intrusion detection problem.

1-2.1 Data Source

An information system typically consists of host machines (e.g., machines running a UNIX operation system and machines running a Windows NT operating system) and communication links connecting those host machines to form a computer network. Two sources of data are widely used to capture events in an information system for intrusion detection: network traffic data and audit trail data (audit data). Network traffic data contain data packets traveling over communication links between host machines to capture events over communication networks. Audit data capture events occurring on a host machine. In this study, we use audit data from a UNIX-based host machine (specifically a Sun SPARC 10 workstation with a Solaris operating system), and focus on intrusions into a host machine that leave trails in computer audit data.

The Solaris operating system from the Sun Microsystems Inc. has a security facility, called BSM. BSM monitors audit events on a host machine. There are over 250 different types of BSM auditable events, depending on the version of the Solaris operating system. Since there are about 284 different types of BSM audit events on our host machine, we consider 284 event types in this study. A BSM audit record for each event contains a variety of information, including the time of the event, event type, user ID, group ID, process ID, session ID, the system object accessed, and so on. In this study, we are interested in the intensity of events. Hence, only the time of audit events is extracted from the audit data and used for intrusion detection. The time unit is second.

1-2.2 Training and Testing Data

In this study, we obtain a collection of audit data for normal events from Massachusetts Institute of Technology - Lincoln Laboratory (MIT-LL). Those events are generated by simulating activities observed in a real computer network system in a normal operation condition. The audit data contain a sequence of 3019 audit events lasting 580 seconds. The intensity of the 3019 audit events ranges from 0 to 135 events per second. We use the first half of the audit data, consisting of 1613 audit events lasting 381 seconds, as the training data. The second half of the audit data, consisting of 1406 audit events lasting 199 seconds, is used for testing. Hence, the number of the normal events in the training data set is similar to the number of the normal events in the testing data set.

The data set of the 3019 audit events is divided into two halves (one for training and another for testing) based on the number of audit events not the time, because we have developed other intrusion detection techniques which obtain their observations from the audit data based on the type of individual events not the time of these events. We want to test these different intrusion detection techniques on the same training data and the same testing data for comparison.

Added to the 1406 normal events in the testing data set are 10107 intrusive events that are generated by simulating a denial-of-service attack. These 10107 intrusive events lasting 10 seconds are generated by simulating events at a much higher level of the event intensity than that of the normal events. The simulation of the 10107 audit events is based on a normal distribution of the event intensity with the mean of 1000 events per second and the standard deviation of 50 events per second. These 10107 intrusive events are inserted in the middle of the 1406 normal events. Hence, the testing data contain totally 11513 audit events with three segments of data in the sequence: 703 normal events (the first half of the 1406 normal events), 10107 intrusive events, and 703 normal events (the second half of the 1406 normal events). The time gap from the last event in one segment of audit data to the first event in the next segment is 1 second.

1-2.3 Observation of the Event Intensity

The event intensity can be measured by counting the number of events per second. We denote this measure of the event intensity as *k(i)*. For example, the training data consist of 1613 audit events for 381 seconds, and produce a sequence of 381 observations of the event intensity (the number of events per second) as shown in Figure 1-1. The testing data consist of 11513 audit events for 209 seconds, and produce a sequence of 209 observations of the event intensity as shown in Figure 1-2. Figure 1-1 and Figure 1-2 also show the autocorrelation of the observations on the event intensity from the audit data.

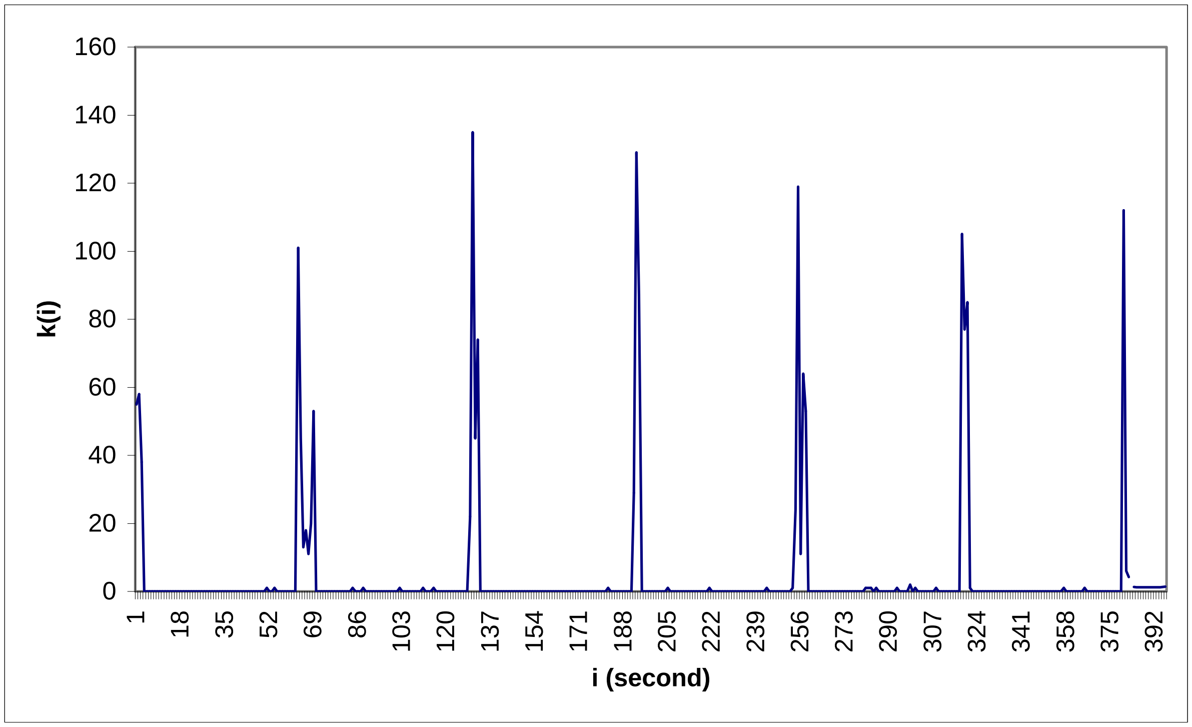


Figure 1-1. The observations of the event intensity, k(i), from the training data using the event intensity method.

In this study, we use the smoothed event intensity by smoothing the observations of the event intensity in the recent past as follows:

$$x(i) = \eta * k(i) + (1-\eta)x(i-1), \qquad (9)$$

where *x(i)* is the smoothed event intensity at time *i*, *k(i)* is the event intensity at time *i*, and η is the smoothing constant. A detailed description of the exponential smoothing method as can be found in [33]. The smoothing is to reduce the effect of outliers or wild values in the observations of the event intensity [33].

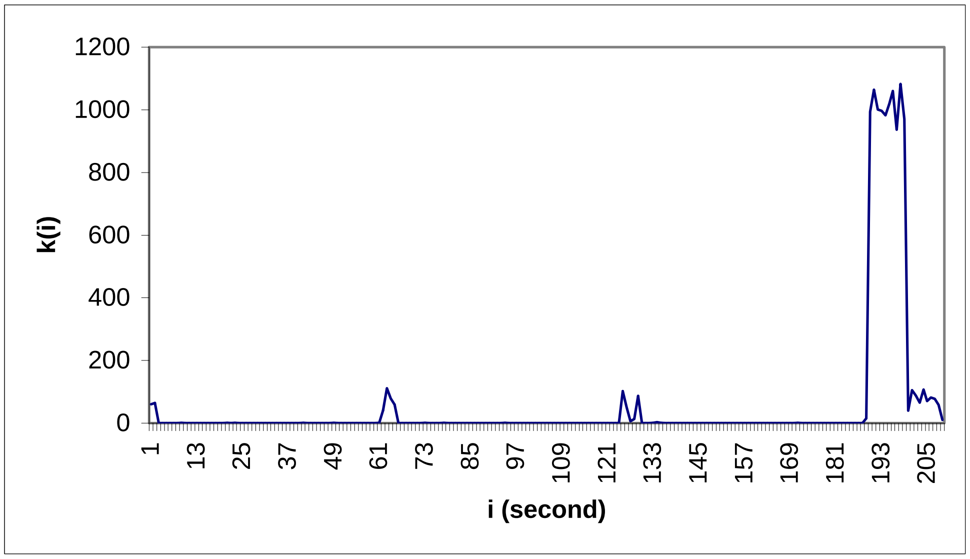


Figure 1-2. The observations of the event intensity, k(i), from the testing data using the event intensity method.

The smoothing constant determines the decay rate or the aging weight of the past observations in computing the smoothed event intensity at the present time. The event intensity at the present time *i* receives a weight of η, the observation at time *i-1* receives a weight of *η(1-η)*, and the observation at time *i-p* receives a weight of $\eta(1-\eta)^p$. Hence, if we are interested in a long-term trend of the observations in the past, we should use a small value of η, e.g., 0.0001. If we are interested in the short-term trend of the observations in the recent past, we should use a large value of *η*, e.g., 0.2. In general, if we wish to have an exponential smoothing system that is equivalent to an *N*-period moving average system, *η* is set as follows [28, 33]:

$$\eta = \frac{2}{N+1} \tag{10}$$

In this study, we use the exponential smoothing method to obtain the short-term trend of the observations in the recent past. Hence, we investigate and compare two values of *η*; 0.2 and 0.3.

Events in an information system occur in the interval shorter than one second as shown in Figure 1-1 and figure 1-2. Hence, the measure of the event intensity in terms of the number of events per second is like sampling events every second. However, the data sampling at every second may leave a time gap for intrusive events such as those from a denial-of-service attack to damage the information system before the next data sample is taken, because a denial-of-service program can generate hundreds or even thousands of events automatically within a second. To prevent this, we transform formula (9) into an equivalent form that is used to update the smoothed event intensity at each event j as follows:

$$x(t_j) = \eta * 1 + (1-\eta)^{(t_j - t_{j-1})} x(t_{j-1}), \quad (11)$$

where t_j is the time when the j^{th} event occurs, and $x(t_j)$ is the event intensity at time t_j.

Table 1-1. The computation of the smoothed event intensity for η = 0.2.

Time t_j	Event j	$x(t_j)$ by Formula (11)	$x(t_j)$ by Formula (9)
0	none	4.0 (arbitrary initial value)	4.0 (arbitrary initial value)
1	1st	$0.2*1+(1-0.2)^{(1-0)}*4.0 = 3.4$	
same as above	2nd	$0.2*1+(1-0.2)^{(1-1)}*3.4 = 3.6$	
same as above	3rd	$0.2*1+(1-0.2)^{(1-1)}*3.6 = 3.8$	$0.2*3+(1-0.2)*4.0 = 3.8$
2	4th	$0.2*1+(1-0.2)^{(2-1)}*3.8 = 3.24$	
same as above	5th	$0.2*1+(1-0.2)^{(2-2)}*3.24 = 3.44$	$0.2*2+(1-0.2)*3.8 = 3.44$
3	none		$0.2*0+(1-0.2)*3.44 = 2.752$
4	none		$0.2*0+(1-0.2)*2.752 = 2.2016$
5	6th	$0.2*1+(1-0.2)^{(5-2)}*3.44 = 1.96128$	$0.2*1+(1-0.2)*2.2016 = 1.96128$

Table 1-1 shows the computation of the smoothed event intensity according to formula (11) in comparison with the computation of the smoothed event intensity according to formula (9) for a given sequence of events. As can be seen from Table 1-1, formula (9) updates the smoothed event intensity for every second, whereas formula (11) updates the smoothed event intensity for every event but produces the same result of the smoothed event intensity at the end of every second.

1-2.4 Application of EWMA Techniques to Intrusion Detection

Both EWMA for autocorrelated data and EWMA for uncorrelated data are tested to compare their performance for intrusion detection.

The application of the EWMA technique for autocorrelated data to intrusion detection takes the following steps.

1. Training. Since the training data set consists of 1613 normal events for 381 seconds, we first obtain the sequence of 381 observations of the event intensity by computing the number of events for each second. The average event intensity of these 381 observations is calculated to become the initial value of the smoothed event intensity, x(0), in formula (11), and the initial value of the EWMA statistic, z(0), in formula (1). Then for each audit event *i* in the training data set, we use formula (11) to obtain the observation of the smoothed event intensity for that event, *x(i)*, and use formula (1) to compute the EWMA statistic for that event, *z(i)*. Finally, the average of the 1613 values of the smoothed event intensity, *x(i),* is computed and used as the initial value of *z(0)* for the testing data. The smoothed event intensity for the last event in the training data set, *x(1613),* is used as the initial value of the smoothed event intensity, *x(0),* for the testing data. For the 1613 events in the training data set, the sum of the squared one-step-ahead prediction errors divided by 1613 is used as the initial value of the smoothed variance of the prediction errors, $\sigma_e(0)^2$, for the testing data.
2. Testing. For each audit event *i* in the testing data set, we first use formula (11) to obtain the smoothed event intensity, *x(i),* formula (1) to obtain the EWMA statistic, *z(i),* formula (8) to compute the estimated variance of the prediction errors, $\sigma_e(i)^2$, and formula (7) to compute the upper and lower control limits, $UCL_x(i)$ and $LCL_x(i)$. Then we evaluate if *x(i)*

falls in [$LCL_x(i)$, $UCL_x(i)$]. If not, an alarm signal is produced on this event; otherwise, no signal is produced.

Table 1-2. Different values for parameters used in the EWMA technique for autocorrelated data.

Smoothing Constant, η	λ for EWMA Statistic	α for Smoothed Variance	L
0.2	0.85	0.05	1.96
		0.001	1.96
		0.0001	1.96
	0.05	0.05	1.96
			3
	0.001	0.001	1.96
	0.0001	0.0001	1.96
	0.00001	0.00001	1.96
0.3	0.05	0.05	1.96
	0.001	0.001	1.96
	0.0001	0.0001	1.96

Table 1-2 shows different values for the parameters in the above formulas that we investigate for EWMA for autocorrelated data. The reason for using the value of 0.2 for the smoothing constant η in formula (11) is given in Section 1-3. In [28], it is suggested that the value of the parameter λ for the EWMA statistic in formula (1) is chosen to minimize the sum of squared one-step-ahead predictions errors on the training data. This method yields the λ value of 0.85 in Table 1-1. For this λ value, several values of α, 0.05, 0.001 and 0.0001, for the smoothed variance in formula (8) are investigated for comparison. In addition to the λ value of 0.85, the λ values of 0.05, 0.001, 0.0001 and 0.00001 are also investigated for comparison. For each of these additional λ values, the value of α is set to the same level as the λ value (0.05, 0.001, 0.0001, or 0.00001).

In addition to the η value of 0.2, the η value of 0.3 along with similar values of λ and α to those for the η value of 0.2 is also investigated for comparison. The value of L is set to 1.96 for all the combinations of parameter values. In addition, the L value of 3 is also investigated for the η value of 0.2. The λ value of 0.05 and the α value of 0.05 yield poor performance as shown in the next section. Hence, the L value is increased from 1.96 to 3 to see if this change of the L value makes any performance difference.

The application of the EWMA technique for uncorrelated data to intrusion detection takes the following steps.

1. Training. Since the training data set consists of 1613 normal events for 381 seconds, we first obtain the sequence of 381 observations of the event intensity by computing the number of events for each second. The average event intensity of these 381 observations is calculated to become the initial value of the smoothed event intensity, x(0), in formula (11), and the initial value of the EWMA statistic, z(0), in formula (1). Then for each audit event *i* in the training data set, we use formula (11) to obtain the observation of the smoothed event intensity for that event, *x(i),* and use formula (1) to compute the EWMA statistic for that event, *z(i)*. Finally, the average of the 1613 values of the smoothed event intensity, *x(i),* is computed and used as the initial value of *z(0)* for the testing data. This average is also used as the estimated μ_x in formulas (2) during testing. The smoothed event intensity for the last event in the training data set, *x(1613),* is used as the initial value of the smoothed event intensity, *x(0),* for the testing data. The standard deviation of *x(i)* for the training data is used as the estimated σ_x in formula (3) during testing. Formulas (2) and (3) are used to compute μ_z and σ_z. Formula (4) is then used to compute the upper and lower control limits, UCL_z and LCL_z, which are used during testing.

2. Testing. For each audit event *i* in the testing data set, we first use formula (11) to obtain the smoothed event intensity, *x(i),* and use formula (1) to obtain the EWMA statistic, *z(i).*

Then we evaluate if *z(i)* falls in [LCL_z, UCL_z]. If not, an alarm signal is produced on this event; otherwise, no signal is produced.

As discussed in Section 1-2.3 and shown in Figure 1-1 and Figure 1-2, the data of the event intensity in the information system are autocorrelated. The data of the smoothed event intensity are also autocorrelated because of the smoothing method. Hence, the EWMA technique for uncorrelated data is tested to compare with the EWMA technique for autocorrelated data only for the η value of 0.2 and the λ value of 0.0001 that yield the best performance for the EWMA technique for autocorrelated data among all the parameter combinations as shown in the next section. The value of L in formula (4) is also set to 1.96 at the significance level of 0.05.

1-3 Results and Discussions

The testing data consist of 703 normal events, 10107 intrusive events, and 703 normal events in sequence. For each EWMA technique with a given parameter combination, we compute the number of the false alarms and the number of hits. A false alarm is a signal on a normal event. A hit is a signal on an intrusive event. We also check how soon the intrusion is detected by noting the first intrusive event with a signal. Table 1-3 shows the number of the hits, the number of the false alarms, and the first signaled intrusion event for the two EWMA techniques with different combinations of parameter values.

The examination on the performance of the EWMA technique for autocorrelated data leads to the following findings.

1. As can be seen from Table 1-3, among all the parameter settings for the η value of 0.2, the λ value of 0.0001 and the α value of 0.0001 (highlighted by * in Table 1-3) produce the best performance with the 5392 hits out of the 10107 intrusive events, the 0 false

alarm out of the 1406 normal events, an early detection at the 958th intrusion event. Figure 1-3 shows more performance details for this parameter setting.

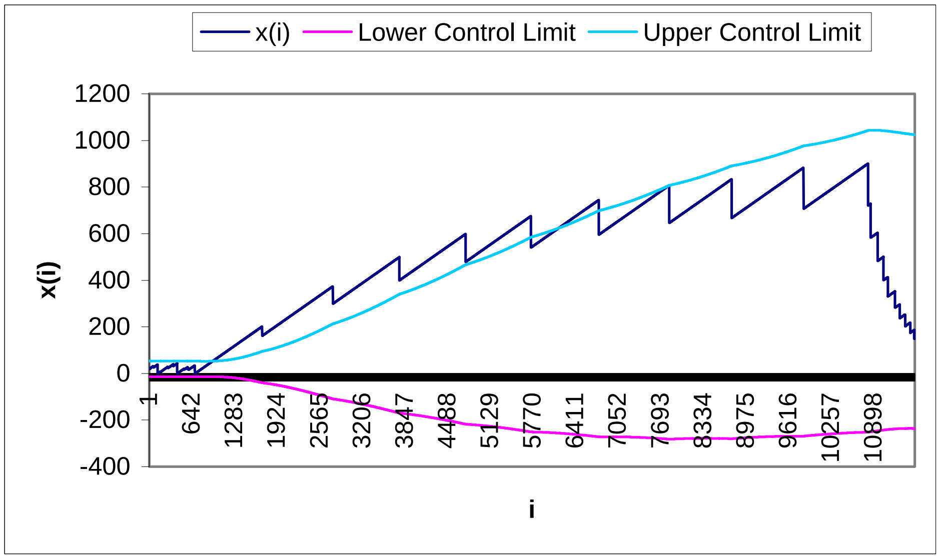
2. For this parameter setting (the λ value of 0.0001 and the α value of 0.0001), signals are not produced for all the 10107 intrusive events. Signals are produced only for those early intrusive events. As the center line and the control limits of the control chart gradually adjust to the smoothed intensity level of the intrusive events, no signals are produced on the later intrusive events. This is acceptable because the early detection of the intrusive events should already trigger actions to stop the later intrusive events. The early detection of the intrusion events is more important than the 100% hit rate for all the intrusive events.
3. For this parameter setting (the λ value of 0.0001 and the α value of 0.0001), changing the η value from 0.2 to 0.3 does not make much performance difference.
4. Larger values of λ, such as 0.05 and 0.85 (the optimal value for the time-series data of the smoothed event intensity), produce smaller prediction errors, thereby resulting in smaller estimated σ_e and smaller in-control ranges as shown in Figure 1-4. This produces the false alarms on both the first 703 normal events and the last 703 normal events. Even increasing the value of L from 1.96 to 3 does not help much in overcoming this difficulty as shown in Table 1-3. When η represents the short-term trend of the event intensity in the recent past, λ and α as the parameters in determining the control limits should reflect the long-term trend of the event intensity. This is why the parameter setting of the η value of 0.2, the λ value of 0.0001 and the α value of 0.0001 produces the best performance on the smoothed event intensity as stated in Finding 1.

5. If the values of λ and α are too small, such as 0.00001, the center line and the control limits of the EWMA control chart become too sluggish to update the long-term trend of the smoothed event intensity, thereby resulting in the false alarms in the last 703 normal events when the smoothed event intensity drops after the intrusive events as shown in Figure 1-5 and Table 1-3.

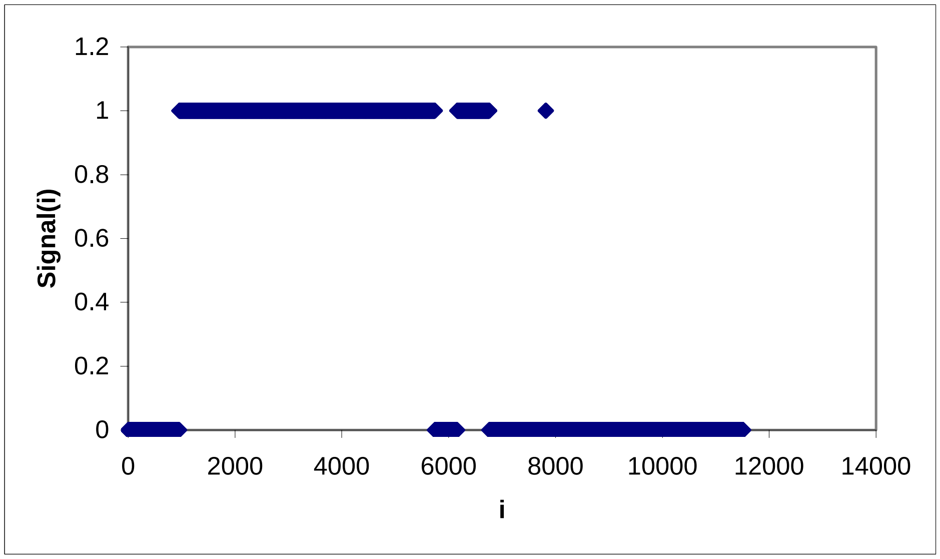
The performance of the EWMA technique for uncorrelated data is shown in Figure 1-6 and Table 1-3. Because the control limits do not change, there are false alarms on all the last 703 normal events whose smoothed event intensity still carries over the high level of the smoothed event intensity of the 10107 intrusive events before these 703 normal events.

In summary, it appears from the testing results that both EWMA for autocorrelated data and EWMA for uncorrelated data can work well for detecting intrusions that manifest themselves through significant changes in the intensity of events occurring in an information system. The advantage of the EWMA technique for uncorrelated data is that it can detect not only abrupt changes in the event intensity but also small mean shifts through the gradually increased or decreased event intensity as discussed in [25]. However, if the EWMA technique for uncorrelated data is used, the initial value of the smoothed event intensity needs to be reset after intrusions are detected for preventing the carry-over effect.

If the EWMA technique for autocorrelated data is used, the reset of the initial value of the smoothed event intensity is not necessary as the EWMA technique for autocorrelated data automatically adjusts the control limits to account for the carry-over effect. Overall, the smoothing constant for computing the smoothed event intensity should not be too small in order to capture the short-term trend of the event intensity in the recent past. The parameters λ and α for setting the control limits to reflect the long-term trend of the smoothed event intensity should be much smaller than the smoothing constant. The L value for the significance level of 0.05 appears to work well. The EWMA technique for autocorrelated data may not be able to detect small mean shifts through the gradually increased or decreased event intensity.

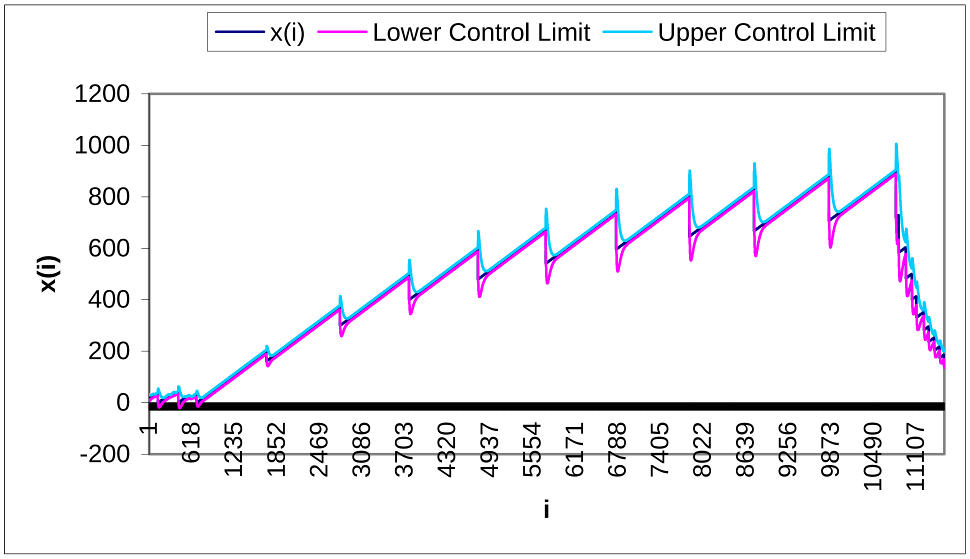


a. The plot of x(i), LCL(i), and UCL(i).

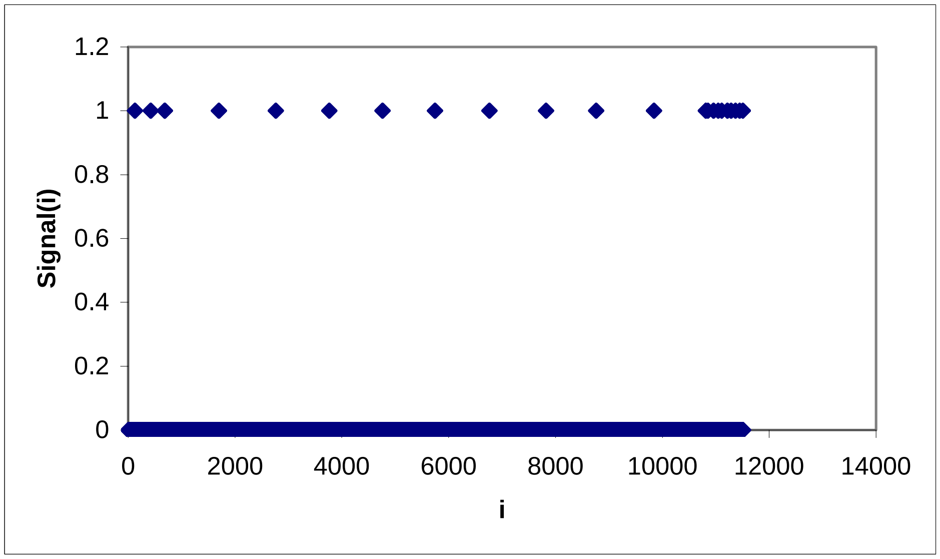


b. The plot of signal(i).

Figure 1-3. The performance of EWMA for autocorrelated data with the η value of 0.2, the λ value of 0.0001, the α value of 0.0001, and the L value of 1.96.

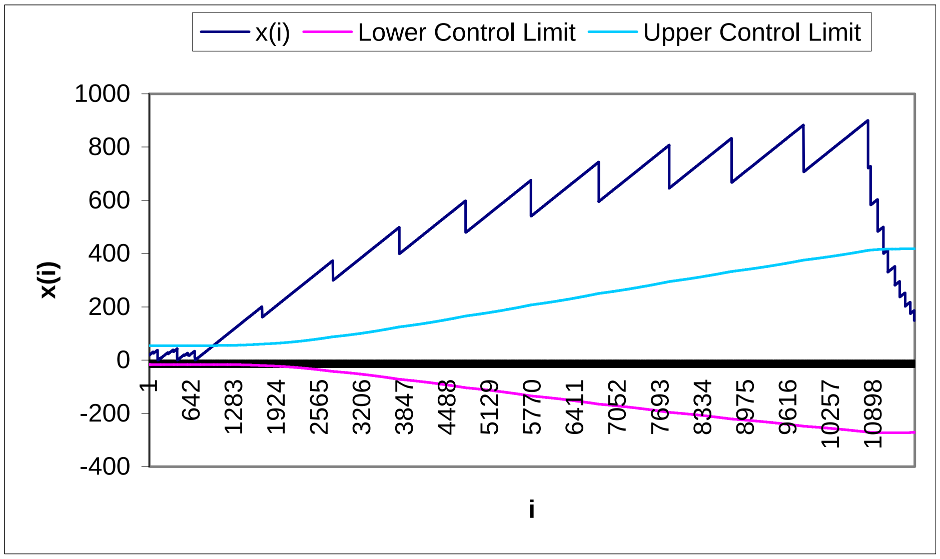


a. The plot of x(i), LCL(i), and UCL(i).

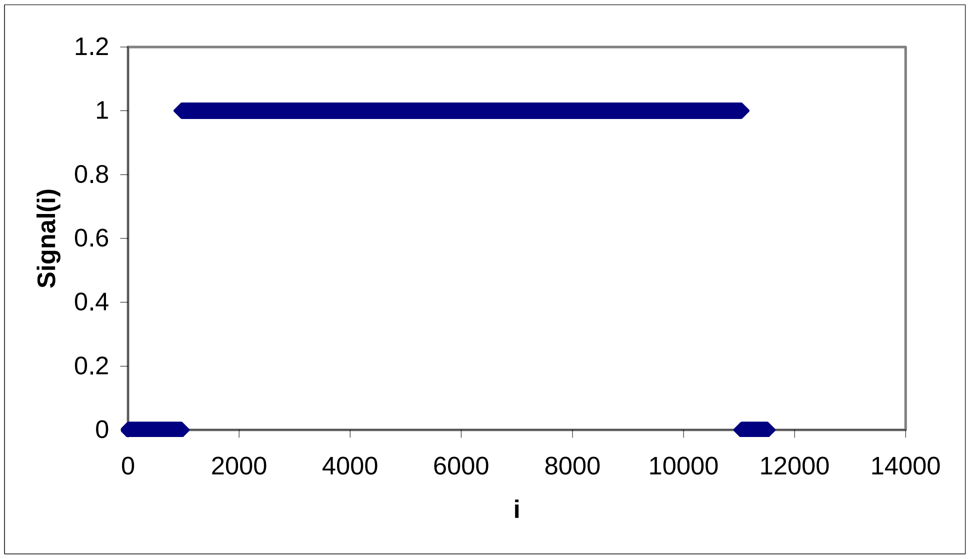


b. The plot of signal(i).

Figure 1-4. The performance of EWMA for autocorrelated data with the η value of 0.2, the λ value of 0.05, the α value of 0.05, and the L value of 1.96.

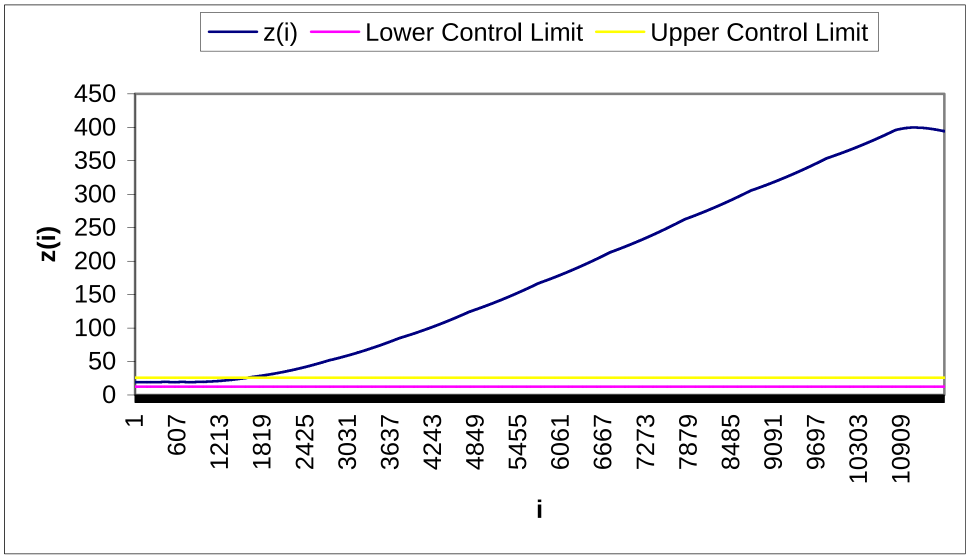


a. The plot of x(i), LCL(i), and UCL(i).

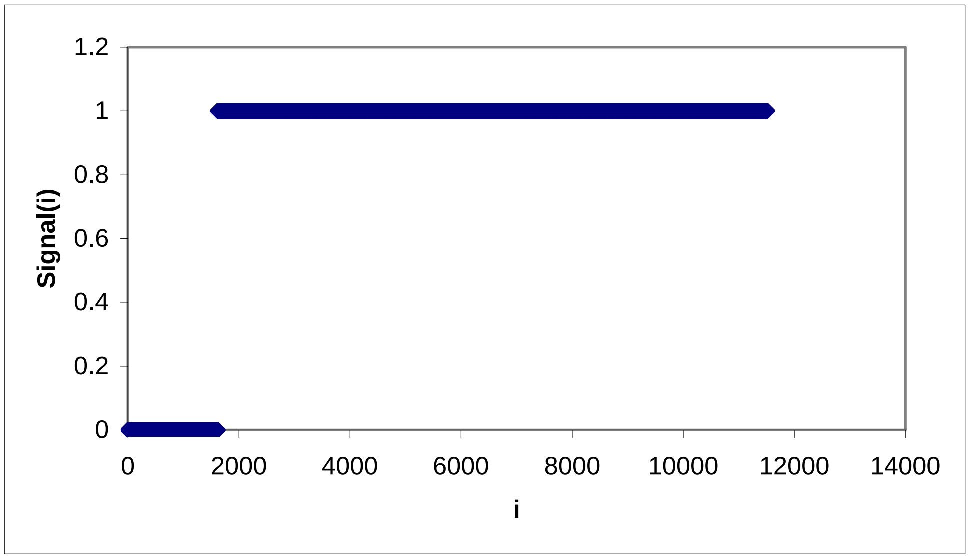


b. The plot of signal(i).

Figure 1-5. The performance of EWMA for autocorrelated data with the η value of 0.2, the λ value of 0.00001, the α value of 0.00001, and the L value of 1.96.



a. The plot of x(i), LCL(i), and UCL(i).



b. The plot of signal(i).

Figure 1-6. The performance of EWMA for uncorrelated data with the η value of 0.2, the λ value of 0.0001, and the L value of 1.96.

Table 1-3. Performance results of the two EWMA techniques with different combinations of parameter values.

EWMA	η	λ	α	L	Hits (from 10107 intrusive events)	False Alarms (from 1406 normal events)	First Signaled Intrusive Event
Autocorrelated Data	0.2	0.85	0.05	1.96	9	21 (in F & L)	1700th
			0.001	1.96	18	25 (in F & L)	1700th
			0.0001	1.96	18	27 (in F & L)	1700th
		0.05	0.05	1.96	44	60 (in F & L)	1700th
				3	18	27 (in F & L)	1700th
		0.001	0.001	1.96	655	437 (in L)	917th
		0.0001*	0.0001*	1.96*	5392*	0*	958th*
		0.00001	0.00001	1.96	9846	233 (in L)	967th
	0.3	0.05	0.05	1.96	0	0	N/A
		0.001	0.001	1.96	675	354 (in L)	891th
		0.0001	0.0001	1.96	4789	0	935th
Uncorrelated Data	0.2	0.0001	N/A	1.96	9192	703 (in L)	1621th

Note: F stands for the first 703 normal events in the testing data, and L stands for the last 703 events in the testing data.

Chapter 2: Robustness of the Markov Chain Model for Cyber Attack Detection

2-1. Markov Chain Model

A discrete-time stochastic process can be described as an arbitrary family of random variables $X(t,\xi,\theta)$ where t is a series of discrete time points, $t_0 < t_1 < \cdots < t_i < \cdots < t_n \in T$ and T is a parametric space. Here ξ represents all the random factors of the system, design parameter θ belongs to an infinite set of all possible design parameters, and the set of all possible values assumed by the process, *S*, is called the state space. A realization of the discrete-time stochastic process may be described by plotting $X(t)$ against discrete time $t = t_0, t_1 \cdots t_i \cdots, t_n \in T$ when we assign $\theta \in \Theta$ a value.

A Markov chain is a special type of discrete-time stochastic process with the assumption [34-37]

$$P(X_{t+1} = i_{t+1} \mid X_t = i_t, X_{t-1} = i_{t-1}, \ldots, X_0 = i_0) = P(X_{t+1} = i_{t+1} \mid X_t = i_t), \tag{1}$$

for any $t \in T$ and $i_t \in S$. That is, the probability distribution of the state at time *t+1* depends on the state at time *t*, and does not depend on the previous states leading to the state at time *t*. Therefore, a Markov chain is a first-order Markov model which considers only one-step transition probabilities. A high-order Markov model considers state transitions over more than one time step. If we assume that a state transition from time *t* to time *t*+1 is independent of time, the Markov chain becomes a stationary Markov chain [6] and is denoted

$$P(X_{t+1} = i_{t+1} \mid X_t = i_t) = P(X_{t+1} = j \mid X_t = i) = p_{ij}, \text{ for all } t \text{ and all states,} \tag{2}$$

where p_{ij} is the probability that the system is in a state j at time t+1 given the system is in state i at time t. If the system has a finite number of states, 1, 2, …, s, the Markov chain model can be defined by a transition probability matrix [38]

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1s} \\ p_{21} & p_{22} & \cdots & p_{2s} \\ \vdots & \vdots & \vdots & \vdots \\ p_{s1} & p_{s2} & \cdots & p_{ss} \end{bmatrix}, \tag{3}$$

where

$$\sum_{j=1}^{j=s} p_{ij} = 1. \tag{4}$$

The initial probability distribution [38] is represented by

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_s \end{bmatrix}, \tag{5}$$

where q_i is the probability that the system is in state i at time 0. The probability that a sequence of states X_1, …, X_T occurs in the context of the Markov chain model is computed using

$$P(X_1, \cdots, X_T) = q_{x_1} \prod_{t=2}^{T} P_{X_{t-1}X_t} \tag{6}$$

A logarithmic transformation of Equation (6) can be expressed as

$$\log(P(X_1, \cdots, X_T)) = \log(q_{x_1}) \sum_{t=2}^{T} \log(P_{X_{t-1}X_t}). \tag{7}$$

Since we will be working with a very large sequence of events and we are taking the logarithm of the probabilities the central limit theorem can be applied. According to the Central Limit Theorem, we can conclude that the sequence of probabilities we obtained from the Markov chain model conform to the normal distribution with which we are familiar and is easy for us to handle.

The transition probability matrix and the initial probability distribution of a Markov chain model can be obtained from the observations of the system state in the past. Given the observations of the system state X_0, X_1, X_2, …, X_{N-1}, we estimate the transition probabilities and the initial probabilities using [39-40]

$$p_{ij} = \frac{N_{ij}}{N_{i.}} \tag{8}$$

and

$$q_i = \frac{N_i}{N} \tag{9}$$

where

- N_{ij} is the number of observation pairs, X_t and X_{t+1}, with X_t in state i and X_{t+1} in state j,
- $N_{i.}$ is the number of observation pairs, X_t and X_{t+1}, with X_t in state i and X_{t+1} in any one of the states 1, …, s,
- N_i is the number of X_t's in state i, and
- N is the total number of observations.

Bayes parameter estimation can also be used to estimate the transition probability matrix and the initial probability distribution from historic data. However, Bayes' parameter estimation is not adopted in this study due to its high computational cost. Provided with sufficient historic data, the estimation of the transition probability matrix and the initial probability distribution via formulas (8) and (9) can be quite stable.

2-2. Applications

In this section, we define the attack detection problem, including the data source and problem representation, by way of a particular example. We will further illustrate the robustness properties of the Markov chain technique through a second application.

A. Application 1

A computer and network system within an organization typically includes a number of host machines (e.g., machines running a UNIX operation system and machines running the Windows NT operating system) and communication links connecting those host machines. Currently two sources of data have been widely used to capture activities in a computer and network system for attack detection: network traffic data and audit trail data (audit data). Network traffic data contain data packets traveling over communication links between host machines, and thus capture activities over communication networks. Audit trail data capture activities occurring on a host machine. In this study, we used audit data from a UNIX-based host machine (specifically a Sun SPARC 10 workstation with the Solaris operating system), and focused on attacks on a host machine that left trails in the audit data.

The Solaris operating system from Sun Microsystems Inc. has a security extension called the Basic Security Module (BSM). The BSM extension supports the monitoring of activities on a host by recording security-relevant events. A BSM auditable event falls into one of two categories, kernel events or user-level events. Kernel events are generated by system calls to the kernel of the Solaris operation system while user-level events are generated by application software.

There are more than 250 different types of BSM auditable events, depending on the version of the Solaris operating system. There are also about 284 different types of BSM audit events on the host machine used in this study. A BSM audit record for each event contains a variety of information, including the event type, user ID, group ID, process ID, etc. In this study, we extract and use only the *event type*, one of the most critical characteristics of an audit event. Therefore, activities on a host machine are captured through a continuous stream of audit events, each characterized by the event type.

By considering computer audit data with only event type, we detect attacks on a host machine. When an attack is detected from a series of audit events, we can trace the original audit records for the series of audit events and obtain more information such as the user(s) and process(es) from the original audit records to assist responses to an attack.

Both normal and attack activities on a host machine contain sequences of computer actions with random behavior. Sequences of computer actions induce sequences of audit events. Random behaviors create the uncertainty in predicting the next audit event. Considering the 284 types of audit events as the 284 possible states of a host machine, event sequences on the host machine can be represented as a discrete-time stochastic process. Discrete points in time for the discrete-time stochastic process are defined not by a fixed time interval but by times when audit events take place, because we are mainly interested in event transitions in this study. In chapter 1, we develop a technique that examines the time interval or the event intensity for a given period of time from computer audit data.

For attack detection, we want to build a long-term norm profile of event sequence, and to compare the event sequence in the recent past to the long-term norm profile for detecting a large difference. Using formulas (8) and (9), we train and build a Markov chain model of event sequence as the long-term norm profile by learning the transition probability matrix and the initial probability distribution from a stream of audit events that is observed during the normal usage of the host machine.

We define the event sequence in the recent past by opening up an observation window of size *N* on the continuous stream of audit events to view the last *N* audit events from the current time *t*, $E_{t-(N-1)}$, …, E_t, where E represents an event. At the next time *t+1*, the observation window contains E_{t-N+2}, …, E_{t+1}. We investigated $N = 100$ and $N = 10$ to determine how the window size affects the robustness of our technique and found that $N = 10$ is superior to a window size of $N = 100$. This phenomenon is due in part by the fact that as the window size increases, the opportunity for more false alarms would also increase; thus, diminishing the performance of the Markov chain technique.

For the audit events E_{t-99}, …, E_t in the window at time *t*, we examine the type of each audit event and obtain the sequence of states X_{t-99}, …, X_t appearing in the window, where X_i is the state (the type of audit event) that the audit event E_i takes. Using formula (6), we compute the probability that the sequence of states X_{t-99}, …, X_t occurs in the context of the normal usage; that is, the probability that the Markov chain model of the norm profile supports the sequence of states X_{t-99}, …, X_t,

$$P(X_{t-99}, \cdots, X_t) = q_{X_{t-99}} \prod_{i=t-98}^{t} p_{X_{i-1}X_i}$$

The larger the probability obtained, the more likely the sequence of states results from normal activities. A sequence of states from attack activities is expected to receive a low probability of support from the Markov chain model of the norm profile.

It is possible that a sequence of states from a window of the testing data presents an initial state and/or some state transitions which are not encountered in the training and thus have the probabilities of zero in the initial probability distribution or the transition probability matrix of the Markov chain model. While using formula (6) to infer the probability of support to the sequence of states, the probabilities of zero would dominate the final probability result from formula (6) and make it zero. This would make it impossible to distinguish attack activities from normal activities with noises, since noises in normal activities may introduce a new initial state and/or new state transitions. The amount of the new initial state and/or new state transitions from noises in normal activities is expected to be much less than the amount of the new initial state and/or new state transitions from attack activities. Formula (7) is used to get the log-probability of an event sequence.

If we observe the host machine for a very long period of time, we have the opportunity to observe every possible initial state and state transition due to random noises in behavior. However, not every possibility appeared in our training data due to the limited sample size of the data. For example, if the true probability of an initial state is 0.00001 and the sample size of the training data is 1613, the expected number of times that this initial state appears in the training data [6] becomes 0.01613 (equal to 0.00001*1613). That is, we should not expect to observe this initial state in the training data, although the true probability of this initial state is not zero.

Therefore, while using formula (6) to infer the probability of support to a sequence of states, we replace the probability of zero for a new initial state or a new state transition with a small probability value. By assigning a small probability rather than zero to the new initial state or the new state transition, the small noise effects in normal activities on the final probability result of formula (6) becomes distinguishable from the larger effect of attack activities on the final probability result of formula (6).

In this study, the sample size of the training data (to be presented in the next section) is 1613. Since 1/1613 yields 0.0006, any initial state or state transition with a true probability less than 0.0006 is not expected to appear in the training data. Therefore, any small probability values less than 0.0006 can be chosen. A smaller probability value is better in magnifying the difference in the amount of the new initial state and new state transitions in attack activities than those resulting from noises in normal activities. In this study, we assign the probability value of 0.00001 to an initial state or a state transition, which does not appear in the training data, while using formula (6) to infer the probability of support to a sequence of states. This replacement of zero with a small probability value takes place during the inference and after the estimation of the Markov chain model from the training data is complete.

The learning and inference of the Markov chain model for attack detection are implemented using C++. In our C++ program, we use the data type "double" for the variable that keeps the probability of support to the sequence of states from a window. A variable of the data type "double" takes 64 bit of memory with the value range [1.0E-323, 1.0E+323] in scientific expression. For probability values smaller than 1.0E-323, the variable is assigned the value of zero. The results and discussion of this application are provided in Section 2-4.

B. Application 2

A second application is presented in this section to further demonstrate the Markov chain technique using a larger dataset. In this situation we have two large datasets, a Mill dataset and a Pascal dataset, which were obtained from the MIT Lincoln Laboratory. Mill and Pascal are the names of the host machines of the network constructed by the MIT Lincoln Laboratory in order to simulate the environment of the network in the real world and thus provide a test bed of comprehensive evaluations for various intrusion detection systems. Our Mill and Pascal audit datasets are collected from these host machines. The operating systems are Solaris 2.5.1 and Solaris 2.7, respectively. The Mill dataset includes 15 normal sessions consisting of 68872 audit events and seven attack sessions. The Pascal dataset includes 63 normal sessions consisting of 81756 audit events and four attack sessions.

For our Markov chain technique, we use the data collected in the last hour from the machine named Mill, which only contained the normal audit data, as the training dataset. The dataset collected in all three hours from the same machine was used as the testing dataset. The data for the machine named Pascal was collected identically to that of the Mill machine. The results and discussion of this application are provided in Section 2-4.

2-3. Training and Testing Data

In this section, we present the training and testing data used in the example outlined in Section 2-2 involving four datasets with various levels of noise intrusion. Audit data of normal activities are required for estimating a Markov chain model of the norm profile. We use a sample of audit data for normal activities from the MIT Lincoln Lab, containing a stream of 3019 audit events [2] for our study. Computer audit data for normal activities from the MIT Lincoln Lab are generated by simulating activities in a real computer and network system. This sample of audit data for normal activities, consisting of 2316 audit events, is divided into two parts; one part consisting of 1613 audit events and the other part consisting of the remaining 703 audit events. The part consisting of 1613 audit events is used for training a Markov chain model of the norm profile. The second part of the audit data, consisting of 703 audit events, is used for testing.

For testing, audit data of attack activities are generated in our lab by simulating 15 attack scenarios, in a random order, which have been collected over several years from various information sources. Some examples of the attack scenarios are password-guessing, attempts to gain an unauthorized access remotely, etc. The attack scenarios are manually run on the host machine while the auditing facility is turned on, resulting in a stream of 1225 audit events. As a result, the training data consist of the 1613 audit events for normal activities from the MIT Lincoln Lab, and the testing data consisting of the 703 audit events for normal activities from the MIT Lincoln Lab and the 1225 audit events from attack activities from the simulation in our lab. Although there are two sources for obtaining data – MIT Lincoln Lab and our lab – the same operating system is used at both labs. The characteristic under study, event type, is also the same for both locations.

Next, a Markov chain model of the norm profile is obtained using the training data that contain the 1613 audit events for normal activities. As will be discussed in the next section, there are 284 possible types of audit events, of which only 11 types of audit events will appear in the training data. With 11 types of audit events, there are at most 132 parameters (equal to 121 parameters for the transition probability matrix + 11 parameters for the initial probability distribution) in the Markov chain model to estimate from the 1613 data points. Moreover, not all the 121 possible state transitions appear in the training data. The 1613 data points are sufficient to estimate the parameters in the Markov chain model in this situation.

For testing, the 1225 audit events (t = 0, 1, …, 1224) from attack activities and the 703 audit events (t = 0, 1, …, 702) from normal activities are viewed through a moving window of 100 events, creating 1126 windows (t = 99, 100, …, 1224 or window no. 1-1126) for attack activities and 604 windows (t = 99, 100, …, 702 or window no. 1-604) for normal activities. Each of the first 99 audit events for either attack activities or normal activities does not create a window, because the event and preceding events together do not provide 100 audit events for a complete window. The robustness of the Markov chain technique will be examined for window sizes of $N = 10$ and $N = 100$ and the results compared and contrasted.

Note that intrusions and normal activities occur simultaneously in the information system. In real time, intrusive audit data are mixed with white noises of normal audit data. Thus, in order to investigate the robustness of the Markov chain technique or how the performance of this technique depends on the quality of audit data, we create four sets of testing datasets using the 1225 audit events from attack activities and the 703 audit events from normal activities described above. Each subsequent dataset has an increase in mixture of abnormal audit events with the normal events (i.e., the noise level increases from one dataset to the next).

For testing dataset 1 (Pure), we put the 1225 abnormal audit events behind the 703 normal audit events. Obviously, we do not do any mixing for this set of data. For the testing dataset 2, the abnormal and normal audit events are mixed by dividing the 1225 abnormal audit events evenly into two parts, D_{a1} and D_{a2}, and also divide the 703 normal audit events evenly into two parts, D_{n1} and D_{n2}. We then arrange all parts as follows: D_{n1}, D_{a1}, D_{n2}, D_{a2}. We will refer to dataset 2 as the small noise level (SNL) dataset. For testing dataset 3, the 1225 abnormal audit events are divided into seven parts – not necessarily of the same size, D_{a1}, D_{a2}, D_{a3}, D_{a4}, D_{a5}, D_{a6} and D_{a7}, while the 703 normal audit events are also divided into seven parts, D_{n1}, D_{n2}, D_{n3}, D_{n4}, D_{n5}, D_{n6} and D_{n7}. We then arrange all these parts as follows: D_{n1}, D_{a1}, D_{n2}, D_{a2}, D_{n3}, D_{a3}, D_{n4}, D_{a4}, D_{n5}, D_{a5}, D_{n6}, D_{a6}, D_{n7}, D_{a7}. Dataset 3 will be referred to as the large noise level (LNL) dataset. For the testing dataset 4 (Random), we mixed the 1225 abnormal audit events and 703 normal audit events randomly. From a window at time *t*, a sequence of states $X_{t-99},\ldots, X_t$ is obtained and evaluated against the Markov chain model of the norm profile to yield the probability that the sequence of states is supported by this Markov chain model. A high probability indicates that the sequence of states more likely results from normal activities. The lower the probability we obtain, the less likely the sequence of states is a result of normal activities.

2-4. Results and Discussion

A. Receiver Operating Characteristic (ROC) Curves

For a given test, different signal thresholds lead to different pairs of false alarm rate and hit rate that describe the performance of the test according to signal detection theory [41]. A false alarm would occur when normal activities were signaled as attack activities, and a hit would occur when a signal is observed and the event was an attack activity. The false-alarm rate is given by (number of signals)/(the number of normal activities). Obviously, we prefer that the false-alarm rate be small. For example, assume there are 703 normal audit events investigated and 24 of these events signaled as attack activities. We would say that the false-alarm rate is $\frac{24}{703}$ or 0.0341. Obviously, it is desirable to have the false-alarm rate be as small as possible. The hit rate, on the other hand, is given by (number of signals)/(number of attack activities). In this situation, we would prefer that the hit rate be as large as possible.

A Receiver Operating Characteristic (ROC) curve plots pairs of false-alarm rate and hit rate as points when various signal thresholds are used. For reasons discussed previously, we apply the logarithmic transformation given in (7) before the ROC curve is actually constructed.

For Application 2, event-based and session-based ROC curves are investigated. Session-based signal detection involves grouping a set of events into one “session” and estimating the average number of signals along with the standard deviation of the number of signals. For session-based detection, the signal threshold is initially estimated using $\bar{x} + 3s$, where $\bar{x}$ is the average response and *s* represents the standard deviation of the response over the entire session. Based on normal theory, if the data are normally distributed we should expect approximately 99.7% of the data to fall within three standard deviations of the sample average. If data fall beyond three standard deviations, then we may conclude that these data are unusual (i.e., the result of attack activities). As will be shown, it is not necessary for three standard deviations to be used in calculating the signal threshold. We will provide some results when two standard deviations are employed for estimating the signal threshold.

To illustrate the usefulness of ROC curves in general, consider Figure 2-1. The closer the ROC curve is to the upper-left corner (representing 100% hit rate and 0% false alarm rate), the better the performance of the test. If the probability of support to the testing data from a moving window is greater than the decision threshold, the activities in the moving window are classified as normal activities; otherwise the activities in the moving window are classified as attack activities.

B. Application 1

ROC curves for window sizes of N = 100 and N = 10 are presented in Figure 2-1 and Figure 2-2, respectively. The results for each of the four small datasets described in the previous section are plotted for these window sizes.

Examining the ROC curves in Figure 2-1 and Figure 2-2, it is obvious that the detection performance of the Markov chain technique depends strongly on the quality of audit data. That is, as the noise level increases the performance of the Markov chain decreases. We can also conclude that the Markov chain technique with a smaller window size (N = 10) outperforms the Markov chain technique with a window size of N = 100 as the amount of noise in the system increases.

C. Application 2

The ROC curves for the Mill dataset are given in Figure 2-3 through Figure 2-5. Figure 2-3 represents the Event-Based ROC curves for N = 100 and N = 10. Figures 4 and 5 display the Session-Based ROC curves using two standard deviations and three standard deviations, respectively. Based on these results, the detection performances of the Markov chain techniques with N = 100 and N = 10 are comparable for the event-based situation and the session-based situation with two standard deviations. Notice however, that as the standard deviation increases from two to three, (see Figure 2-5) the Markov chain technique with N = 10 is superior to the technique when N = 100. The superior performance of the technique with a smaller window size is due in part to fewer opportunities for false alarms. A second observation is that the performance of the Markov chain technique with N = 10 decreases as the number of standard deviations increase.

ROC curves are also provided for the Pascal dataset under the same conditions as the Mill dataset. The curves are provided in Figures 2-6 through 2-8. Based on these curves, the Markov chain technique appears to be quite robust to the window size in all three cases and to the number of standard deviations for the session-based cases shown in Figures 2-7 and 2-8. However, note that the false alarm rate is higher when N = 100 than when N = 10 for the session-based situations.

2-5. Conclusions

We conclude that in order to apply the Markov chain technique and other stochastic process techniques to modeling the sequential ordering of events, the quality of activity data needs to be improved. The improvement could be through, for example, noise canceling, if we consider data of normal activities are noises and data of intrusive activities are signals.

Our study has shown that the performance of the Markov chain techniques is not always robust to the window size. As the window size increases, the number of false alarms will also generally increase. Further investigation would reveal if the inclusion of noise cancellation techniques would improve the robustness of the Markov chain technique in relation to the window size.

We have also provided some initial results on the choice of the number of standard deviations when using session-based techniques. Our results indicate that as the number of standard deviations is increased, the poorer the performance of the Markov chain technique. This should not be surprising since increasing the threshold would decrease the number of signals and thus the session signal ratio, making normal sessions less distinctive from attack sessions.

Overall, our study provides some support for the idea that the Markov chain technique may not be as robust as the other intrusion detection methods such as the chi-square technique discussed in [13]. But, if the Markov chain approach is employed in conjunction with noise cancellation techniques, it is suspected that anomaly detection can be greatly improved. Future research will focus on improving the robustness of our Markov chain techniques to noise in the testing audit data.

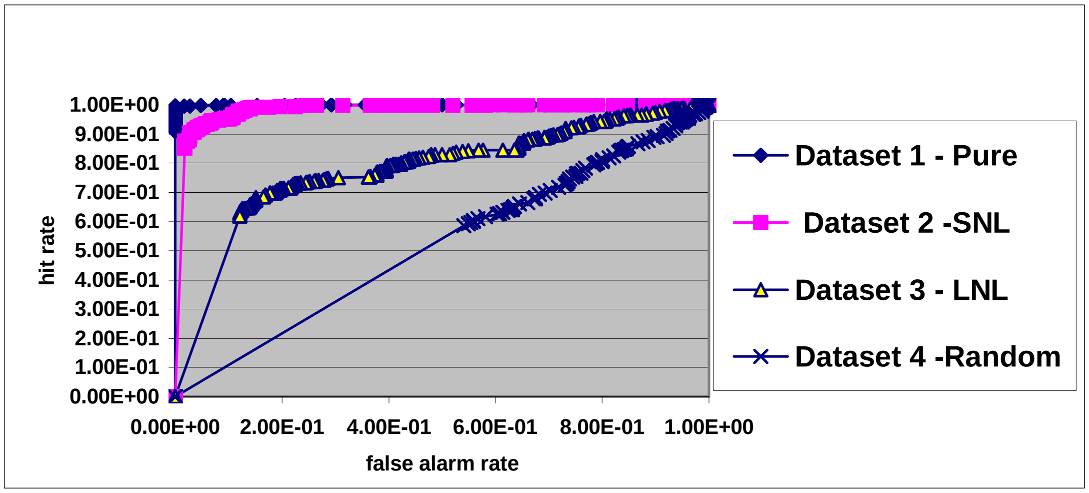


Figure 2-1. ROC Results of the Markov Chain Technique for Four Small Datasets and N = 100

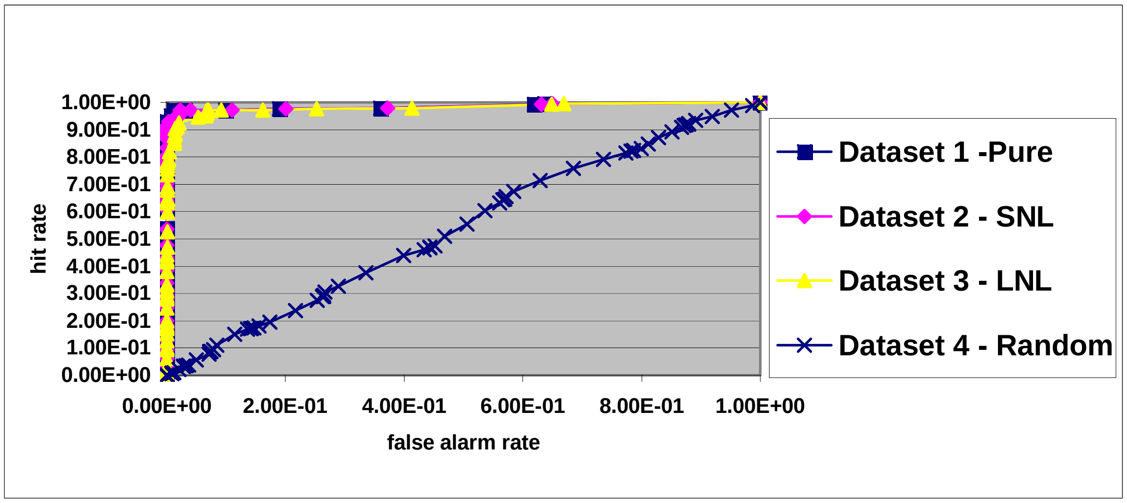


Figure 2-2. ROC Results of the Markov Chain Technique for Four Small Datasets and N = 10

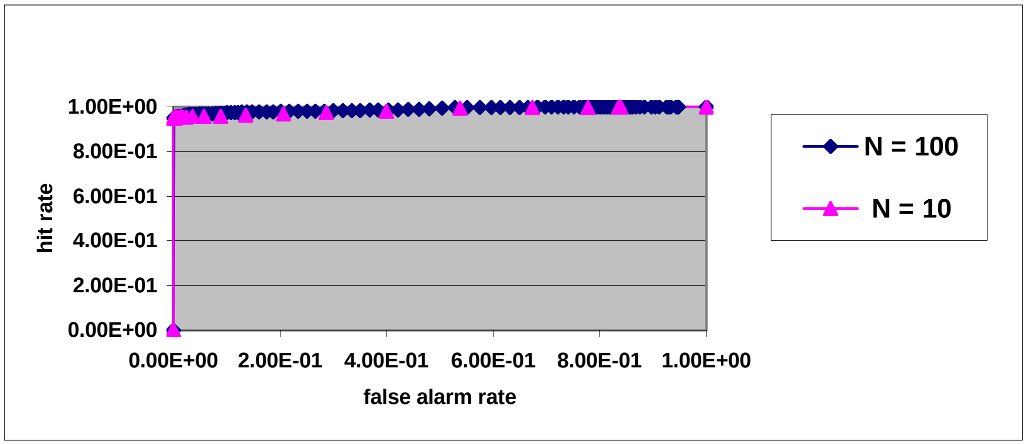


Figure 2-3. Event-Based ROC Curves for the Mill Dataset

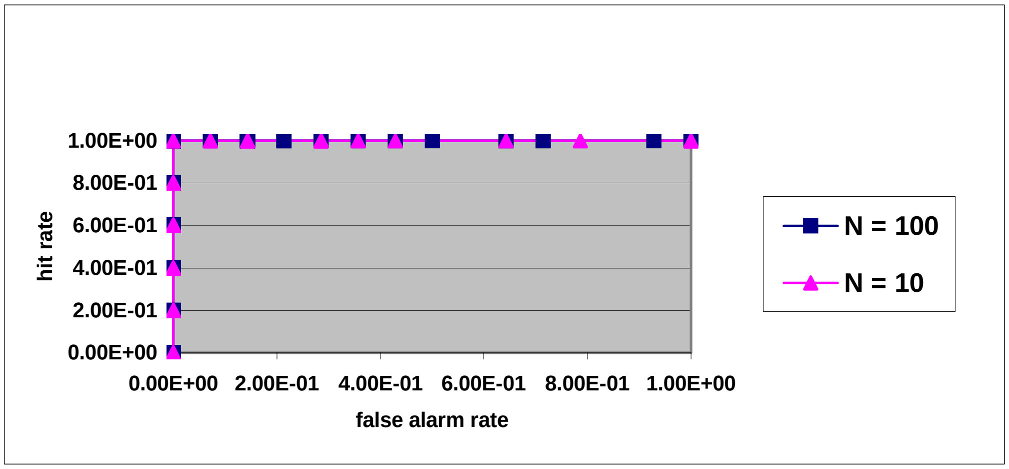


Figure 2-4. Session-Based ROC Curves for the Mill Dataset - Two Standard Deviations

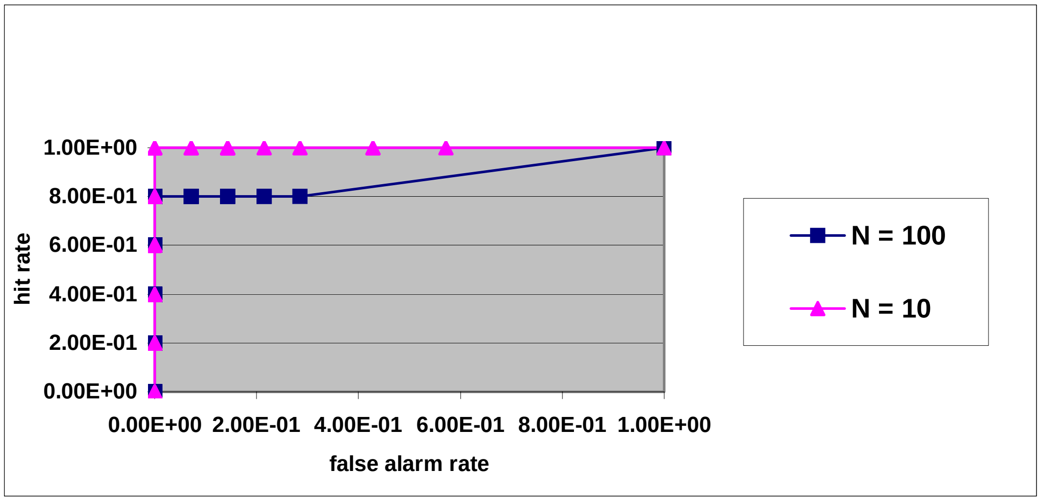


Figure 2-5. Session-based ROC curves for the Mill Dataset - Three Standard Deviations

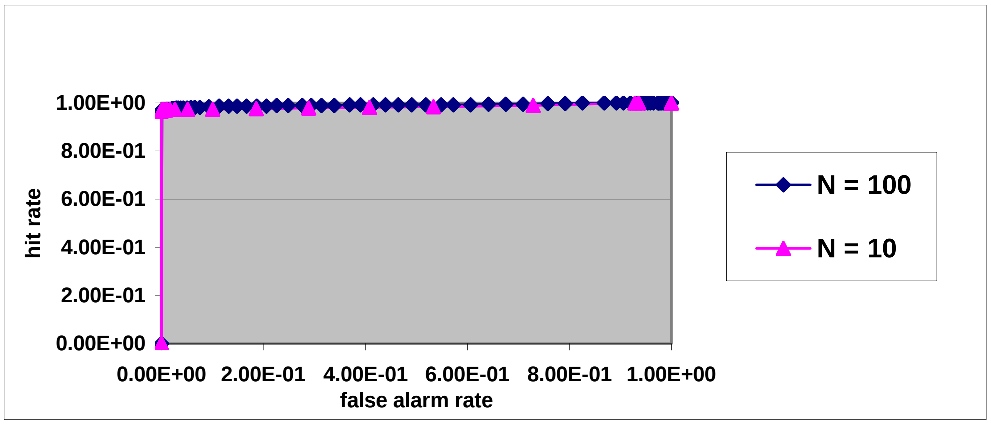


Figure 2-6. Event-Based ROC Curves for the Pascal Dataset

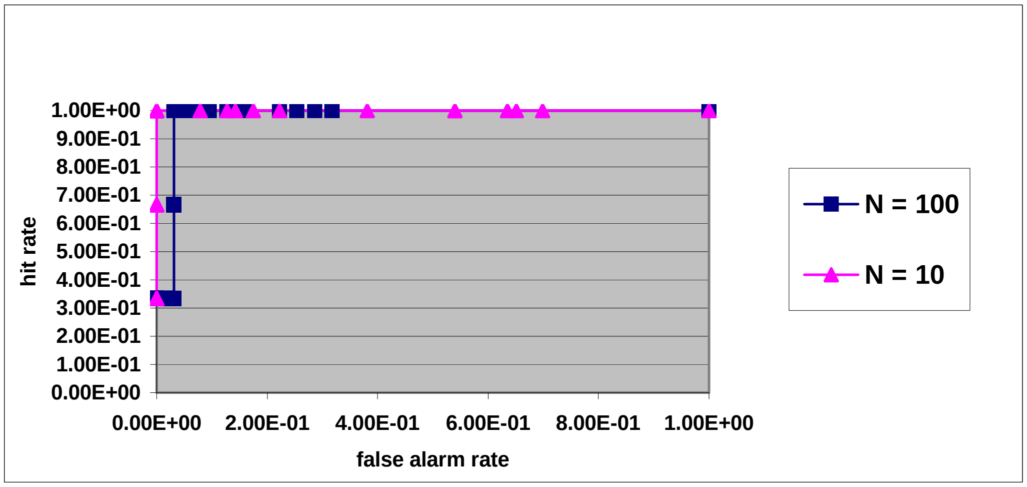


Figure 2-7. Session-Based ROC Curves for the Pascal Dataset - Two Standard Deviations

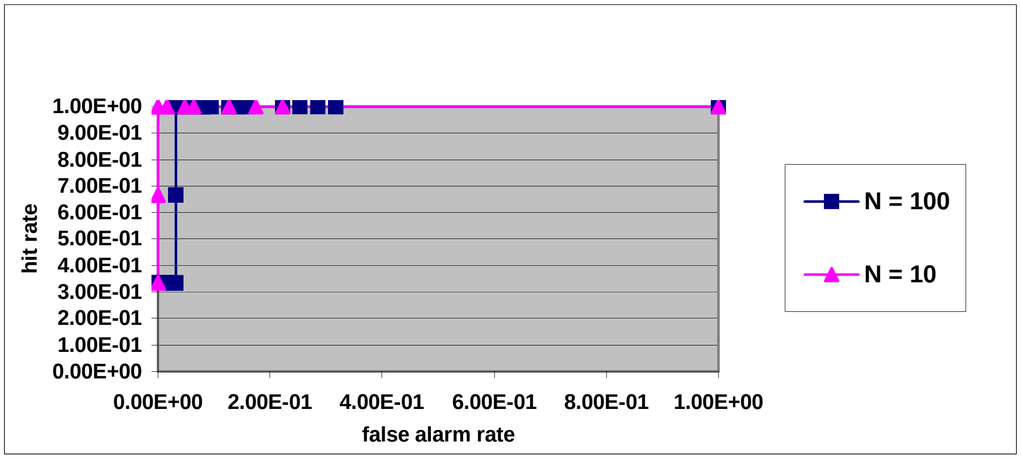


Figure 2-8. Session-Based ROC Curves for the Pascal Dataset - Three Standard Deviations

Chapter 3: Robustness of Chi-square and Canberra Distance Metrics for Computer Intrusion Detection

3-1. Distance Metrics

Hotelling's T^2 test [42] is a common multivariate quality control technique to detect anomalies – deviations from a norm profile - especially mean shifts and counter-relationships in a process measured by multiple variables. Hence, Hotelling's T^2 test can be applied as an anomaly detection technique for intrusion detection. Hotelling's T^2 test uses the following statistical distance to measure the deviation of a *k*-dimensional observation, $\mathbf{x} = [x_1, x_2, \ldots, x_k]'$, from a multivariate normal distribution of the process in control (a norm profile) which is characterized by a sample mean vector, $\bar{\mathbf{x}} = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k})$, and a sample variance-covariance matrix, $\mathbf{S}^{-1} = \frac{1}{n-1}\sum_{j=1}^{n}\left(\mathbf{x} - \bar{\mathbf{x}}\right)\left(\mathbf{x} - \bar{\mathbf{x}}\right)'$:

$$T^2 = \left(\mathbf{x} - \bar{\mathbf{x}}\right)' \mathbf{S}^{-1}\left(\mathbf{x} - \bar{\mathbf{x}}\right) \qquad (1)$$

where *n* is the size of the sample.

Computer audit data of activities in an information system often involve hundreds of variables whose values change frequently. To monitor hundreds of variables from computer audit data at a high frequency of sampling, the computation of the variance-covariance matrix and its inverse in formula (1) requires a large amount of computer memory and computation time, which becomes unacceptable for real-time monitoring and intrusion detection.

In our previous work [12], we introduce the following chi-square distance metric to overcome the scalability problem of the statistical distance metric in Hotelling's T^2 test:

$$\chi^2 = \sum_{i=1}^{k} \frac{\left(x_i - \overline{x_i}\right)^2}{\overline{x_i}} . \tag{2}$$

The chi-square distance measures the deviation from an observation from a norm profile characterized by a sample mean vector only, with the distance scaled simply by the sample mean vector rather than by the complex variance-covariance matrix.

When k is large (e.g., greater than 30), the chi-square distance metric has a normal distribution according to the central limit theorem [43]. Hence, the 3-sigma control limits on the chi-square distance metric can be set to $\left\lfloor \overline{\chi^2} - 3S_{\chi^2}, \overline{\chi^2} + 3S_{\chi^2} \right\rfloor$ for detecting anomalies.

Although the chi-square distance metric can detect only mean shifts, our testing results in [12] demonstrate that the multivariate quality control technique based on the chi-square distance metric produces a comparable performance of intrusion detection to Hotelling's T^2 test based on the statistical distance metric [14]. It is possible that for intrusion detection, mean shifts may be more important than counter-relationships that are not addressed in the chi-square distance metric.

There are also many other distance metrics [44]. We present a few representative distance metrics here. The Euclidean distance between an observation and a sample mean vector **x** is:

$$d(\mathbf{x}, \bar{\mathbf{x}}) = \sqrt{\sum_{i=1}^{k} \left(x_i - \overline{x_i}\right)} = \sqrt{(\mathbf{x} - \bar{\mathbf{x}})'(\mathbf{x} - \bar{\mathbf{x}})} \tag{3}$$

Another distance metric is the Minkowski metric

$$d(\mathbf{x}, \bar{\mathbf{x}}) = \left[\sum_{i=1}^{k} \left| x_i - \overline{x_i} \right|^m \right]^{1/m} \tag{4}$$

For $m = 1$, $d(\mathbf{x}, \bar{\mathbf{x}})$ measures the "city-block" distance between two points in k-dimensions. For $m = 2$, $d(\mathbf{x}, \bar{\mathbf{x}})$ becomes the Euclidean distance. In general, varying m changes the weight given to larger and smaller differences.

Two additional distance metrics are the Canberra metric and the Czekanowski coefficient. Both of them are defined for nonnegative variables only. The Canberra metric and the Czekanowski coefficient are given by equations (5) and (6) respectively.

$$d(\mathbf{x}, \bar{\mathbf{x}}) = \sum_{i=1}^{k} \frac{\left| x_i - \bar{x}_i \right|}{(x_i + \bar{x}_i)} \tag{5}$$

$$d(\mathbf{x}, \bar{\mathbf{x}}) = 1 - \frac{2\sum_{i=1}^{k} \min(x_i, \bar{x}_i)}{\sum_{i=1}^{k} (x_i + \bar{x}_i)} \tag{6}$$

Among these distance metrics, the Euclidean distance metric and the Minkowski distance metric do not scale each dimension when calculating the distance. Our initial testing of the Euclidean distance metric on the same data set as in [12] shows the poor performance of the Euclidean distance metric for intrusion detection. The Canberra distance metric is most similar to the chi-square distance metric in that both of them use the scaled distance and the sample mean vector as one scaling factor in the denominator of formula (2) and formula (5). The Canberra distance adds the observation value as another scaling factor (a part of the denominator). Hence, we carry out this study to compare the performance of the chi-square distance with that of the Canberra distance. We also want to investigate and test the robustness of these distance metrics under different noise levels in data, as noise is typically present in computer audit data collected from an information system (discussed in the next section).

3-2. An Application

This section describes the process of applying the Chi-square and Canberra techniques to intrusion detection. It describes the data source, sets of data used, and problem representation.

3-2.1 Data Source

A computer and network system within an organization typically includes a number of host machines (e.g., machines running a UNIX operation system and machines running the Windows NT operating system) and communication links connecting those host machines. Currently two sources of data have been widely used to capture activities in a computer and network system for attack detection: network traffic data and host audit trail data (audit data). Network traffic data contain data packets traveling over communication links between host machines to capture activities over communication networks. Audit data capture activities occurring on a host machine.

In this study, we use audit data from a UNIX-based host machine (specifically a Sun SPARC 10 workstation with the Solaris operating system), and focus on intrusions into a host machine that leave trails in audit data. The Solaris operating system from the Sun Microsystems Inc. has an auditing facility, called the Basic Security Module (BSM). BSM monitors the events related to the security of a system and records the crossing of instructions executed by the processor in the user space and instructions executed in the kernel. This is based on the assumption that the actions in the user space cannot harm the security of the system and the security-related actions that can impact the system only take place when users request services from the kernel. BSM records the execution of system calls to the kernel by all processes launched by the users and system programs. A full system call trace gives us an overwhelming amount of information, whereas the audit trail provides a limited abstraction of the same information in which the context switches, memory allocation, internal semaphores, and consecutive file reads do not appear. And there is always a straightforward mapping of audit events to system calls.

The BSM audit records contain the detailed information about the audit events in the system. It includes the event type, detailed user and group identification - from the login identity to the one under which the system call is executed, the parameters of the system call execution - file names including full path, command line arguments etc., the return code from the execution, and the error code. However, we use only the event type information contained in it, as many studies [11-18, 20] have shown the effectiveness of such information alone for intrusion detection.

There are 284 different types of events in BSM audit data. In UNIX, there are thousands of commands available. Since the audit events are closer to the core of the operating system, the event type is more representative than the actual command sequences used. For example, we can use any text editor, such as *vi* and *ed* to edit a file, but most of time the audit event stream contains the following event types: AUE_EXECVE, AUE_OPEN_R, AUE_ACCESS, AUE_STAT.

In an intrusion session an intruder tries to hide the not-so-frequently-used commands essential for the intrusion by adding large amount of frequently used commands. The effect of this is that intrusion detection techniques recognize these as noises and fails to detect the intruder's attempts. By using the event type information we are able to extract out the redundant information about which particular commands are used, which particular files are accessed etc., and can focus on which particular kernel-level activities (event types) are being used by the intruder. We also avoid the problem of having inconsistent data when we combine normal and intrusion sessions from different sources such as different host machines. As long as they belong to the same Solaris BSM, we get the consistent event type information.

Intrusions usually occur while normal activities are also occurring on the host machine. If we consider intrusive activities as signals that we want to detect, normal activities are “white noises” in the background. Hence, computer audit data collected from the host machine contain a mixture of normal and intrusive activities - signals plus noises. The number of normal activities is usually greater than the number of intrusive activities. Intrusion detection is to capture signals from noisy data. The robustness of an intrusion detection technique enables the technique to detect signals even when large amounts of noises are present.

3-2.2 Data Sets

In this study we have used four data sets. Among them, the size of the first three data sets is small, containing computer audit data of both normal and intrusive events for a few minutes. The size of the fourth data set is large, containing computer audit data of both normal and intrusive events for four days. We implant different noise levels in the three small data sets to help us in understanding the robustness of the multivariate quality control technique based on two different distance metrics. The large data set resembles a collection of computer audit data for a more realistic mixture of normal and intrusive activities in an information system.

A large portion of data in the four data sets comes from the MIT Lincoln Laboratory (LL) which under the sponsorship of the Defense Advanced Research Projects Agency (DARPA) has created computer audit data of normal and intrusive activities for evaluating intrusion detection systems developed in some DARPA programs [2]. A testbed of a computer and network system is set up at LL. Normal activities are simulated on this testbed to resemble activities in a real-world computer and network system at a US Air Force base. Intrusive activities are also simulated while normal activities are simulated. Intrusive activities are simulated based on a number of intrusion scenarios behind intrusion incidents that have happened in the past. The DARPA-LL evaluation data consists of the three data sets created in 1998, 1999 and 2000 respectively. These three data sets are accessible only to DARPA contractors and not to general public. Hence, a small sample of the 1998 evaluation data set is also provided for general public access. All these data sets include computer audit data of both normal and intrusive activities.

We use computer audit data of only normal events in the small sample of the DARPA-LL 1998 evaluation data as the computer audit data of normal events in the three small data sets. There are 3019 normal events. Since the nature of the intrusive events included in the small sample of the DARPA-LL 1998 evaluation data is not clearly specified, we simulate seven intrusion scenarios in seven sessions respectively based on our own collection of intrusions scenarios that have been used to attack computer and network systems in the past. The simulation of these intrusion scenarios produces computer audit data of 1225 intrusive events used in the three small data sets. The seven intrusion sessions produce 215, 225, 54, 36, 413, 247 and 35 intrusive events respectively. The host machine that we use to simulate the intrusion scenarios has the same Solaris BSM. Hence, having normal data and intrusive data from two different host machines is not a concern in this study as discussed in the previous section.

Among the 3019 normal events, the first 2316 normal events are used as the training data for the multivariate quality control technique to build the norm profile that is characterized by the sample mean vector. The remaining 703 normal events are used as the testing data. All the 1225 intrusive events are used as the testing data. Hence, the training data contains the stream of

2316 normal events, and the testing data contains the stream of 703 normal events and the stream of 1225 intrusive events, totally 1928 events.

To create the three small testing data sets with different noise levels for testing the robustness of the two distance metrics, we arrange the testing data in three different ways. The first testing data set is created by putting the stream of the 703 normal events as the first part of the testing data set and then the stream of 1225 intrusive events as the second part of the testing data set. The second testing data set is created by putting the first 400 normal events of the 703 normal events followed by the 530 intrusive events in the first four intrusions sessions, and then the remaining 303 normal events of the 703 normal events followed by the 695 intrusive events in the remaining three intrusion sessions. Hence, in the second testing data set, the normal and intrusive events are inter-mixed, and the lengths of the normal and intrusion periods are nearly same. The third testing data set is created by putting 100 normal events before each intrusion session. Therefore, the noise level increases from the first testing data test (pure normal data and then pure intrusive data), to the second testing data set, and finally to the third testing data set.

We build the large data set from the DARPA-LL 1998 evaluation data that consist of seven weeks (35 days) of training data and two weeks of testing data. We choose four days of data from this training data set as a representative of these 35 days of data to build the training data and the testing data for the large data set. Only four days of data are chosen to reduce the training and testing time. Two weeks of testing data are not considered because the ground truth (normal or intrusive) of events in these weeks of testing data is not provided. Without the ground truth of these events, we cannot evaluate the detection performance of the multivariate quality control technique based on the two distance metrics.

Table 3-1. Statistics about the large data set.

	Day-1	**Day-2**	**Day-3**	**Day-4**
Event Information				
Number of Events	744085	1320478	2249505	925079
Number of Intrusive Events	3090	36575	16524	31476
Percentage of Intrusive Events	0.42%	2.77%	0.73%	3.40%
Session Information				
Number of Sessions	298	503	339	447
Number of Intrusive Sessions	2	131	29	14
Percentage of Intrusive Sessions	0.67%	26.04%	8.55%	3.13%
Number of Events per Normal Session				
Average	2503	3451	7203	2064
Minimum	1	69	74	96
Maximum	253827	462287	1019443	214705
Number of Events per Intrusion Session				
Average	1545	279	570	2248
Minimum	1101	142	166	1107
Maximum	1989	1737	4986	2841

Two days of data are selected where there are not many intrusive activities, and another two days of data are selected where there are lots of intrusive activities. These four days are week-1, Monday data as day-1 data, week-4, Tuesday data as day-2 data, week-4, Friday data as day-3 data and week-6, Thursday data as day-4 data. Table 3-1 summarizes the statistics about these 4 days of data. As we can see from the table, about 3% of day-2 and day-4 events are intrusive events whereas less than 0.80% of events in day-1 and day-3 data are normal events. In terms of sessions, almost one-fourth of the sessions in day-2 and one-twelfth of the sessions in day-3 are intrusive sessions. Day-1 contains mostly normal sessions, and day-4 also does not have many intrusive sessions. There are totally 176 instances of 9 types of intrusions present in these 4 days of data. Day-1 data and day-2 data are used as the training data, and day-3 and day-4 data are used as the testing data.

3-2.3 Problem Representation

The Basic Security Module (BSM) of the Solaris operating system monitors 284 different types of audit events. An event can only be one of the 284 event types. We need to transform a stream of audit events into a series of observation vectors in the form of $\mathbf{x} = [x_1, x_2, \ldots, x_k]'$. We have 284 variables for 284 types of audit events respectively, producing $\mathbf{x} = [x_1, x_2, \ldots, x_{284}]'$. Each variable represents the smoothed occurrence frequency of the corresponding event type in the recent past. Therefore, an observation vector is a 284-dimensional vector, $\mathbf{x} = [x_1, x_2, \ldots, x_{284}]'$. The exponentially weighted moving average (EWMA) method [25] is used to generate the smoothed occurrence frequency of each event type in the recent past as follows. For the observation value of the nth event in the event stream:

$$\mathbf{x}_n = (x_{1,n}, x_{2,n}, \ldots, x_{284,n}) \tag{7}$$

$$x_{i,n} = \lambda \times \theta + (1-\lambda) \times x_{i,n-1}; \; x_{i,0} = 0, \; 1 \le i \le 284. \tag{8}$$

In formula (7), $\mathbf{x}_n$ is the smoothed observation vector for the nth event. The individual dimensional values are computed using formula (8), where $x_{i,n}$ is the smoothed observation value for event type i for the nth event; θ is an indicator function which is 1 if event type i is present in the *n*th event, and 0 otherwise; $x_{i,n-1}$ is the smoothed observation value at the *(n-1)*th event; and λ is the smoothing constant ($0 < \lambda < 1$). The smoothing constant is usually set to 0.3 [45] which is the value used in this study. Formulas (7) and (8) are used to obtain the observation vector for each audit event in the training data.

The sample mean vector, $\overline{\mathrm{x}} = (\overline{x}_1, \overline{x}_2, ..., \overline{x}_{284})$, is obtained from the series of observation vectors from the training data of normal events to characterize a long-term profile of observation vectors for normal events (norm profile). We use the following incremental updating formula to compute the sample mean vector from the training data:

$$\overline{x}_{i,n} = \frac{(n-1)\overline{x}_{i,n} + x_{i,n}}{n} \text{ where } \overline{x}_{i,0} = 0,\ 1 \le i \le 284 \tag{9}$$

For the observation vector from each audit event in the training data, we use the following formula to compute the chi-square distance metric of this observation vector from the sample mean vector:

$$\chi^2 = \sum_{i=1}^{284} \frac{\left(x_i - \overline{x}_i\right)^2}{\overline{x}_i}. \tag{10}$$

Using all the chi-square distance values from the training data, we compute the average and the standard deviation of the chi-square distance values, $\overline{\chi^2}$ and S_{χ^2}, and set the 3-sigma control limits on the chi-square distance metric to $\left[\overline{\chi^2} - 3S_{\chi^2}, \overline{\chi^2} + 3S_{\chi^2}\right]$ for detecting intrusions in the testing data.

For each audit event in the testing data, we use formulas (7) and (8) to obtain the observation vector and use formula (10) to compute the chi-square distance value. From formula (10), we can see that the chi-square distance value is a positive value. The larger the chi-square distance value, the larger the deviation of this event from the norm profile, and the more likely the event is a part of an intrusion. Hence, only the upper control limit $\overline{\chi^2} + 3S_{\chi^2}$ is used to determine whether to signal the event as intrusive. If the chi-square distance value is greater than or equal to the upper control limit, a signal is produced on this event to claim this audit event as intrusive; otherwise, no signal is produced.

For the multivariate control technique based on the Canberra distance, formula (10) is replaced with the following formula:

$$C = \sum_{i=1}^{284} \frac{\left| x_{i,n} - \overline{x}_i \right|}{\left(x_{i,n} + \overline{x}_i \right)}. \tag{11}$$

The 3-sigma upper control limit is set to $\overline{C} + 3\,S_C$.

3-3 Results and Discussions

This section presents the testing results for the multivariate quality control technique based on the chi-square distance metric and the Canberra distance metric, and discusses the robustness of these distance metrics. We use the Receiver Operating Characteristic (ROC) analysis [46-47] to evaluate the performance of intrusion detection.

For each testing data set, the multivariate quality control technique based on each distance metric produces a set of the distance values (for the chi-square distance metric or the Canberra distance metric) for the audit events in the testing data set. Given a signal threshold, we can compute the false alarm rate and the hit rate. A signal is produced on an event if the distance value for this event is greater than the signal threshold; otherwise, no signal is produced on this event. If a signal is produced on a truly intrusive event, it is a hit. If a signal is produced on a truly normal event, it is a false alarm. Among all the intrusive events in the testing data set, we count how many signals are produced on these events. The hit rate is the ratio of this count to the total number of the intrusive events in the testing data set. Among all the normal events in the testing data set, we count how many signals are produced on these events. The false alarm rate is the ratio of this count to the total number of the normal events in the testing data set. Hence, for a given signal threshold, we obtain a pair of the hit rate and the false alarm rate. By varying the value of the signal threshold, we obtain many pairs of the hit rate and the false alarm rate. These pairs are plotted as a ROC curve. Measuring the hit rate alone indicates only how many intrusive events can be detected. It does not indicate human workload required to analyze false alarms on normal events. A low false alarm rate along with a high hit rate means a desirable performance of intrusion detection in that the detection results can be trusted and human labor required to confirm signals is minimized.

3-3.1 Small Data Sets

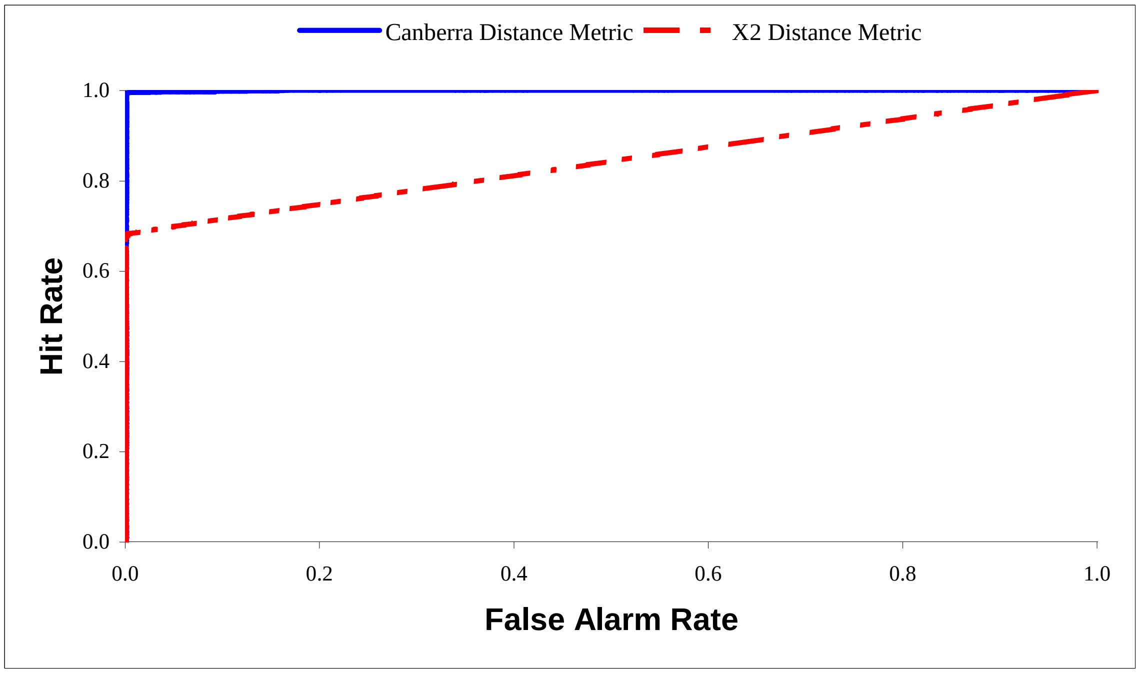


Figure 3-1. ROC curve for the first small data set at the lowest noise level.

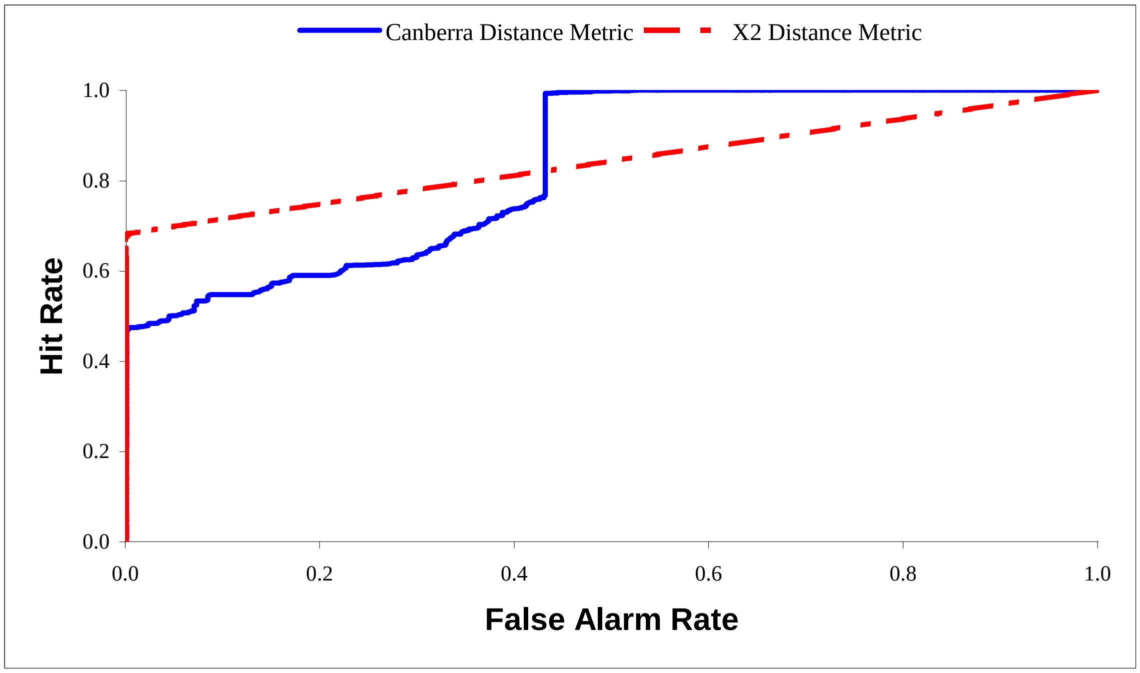


Figure 3-2. ROC curve for the second small data set at the intermediate noise level.

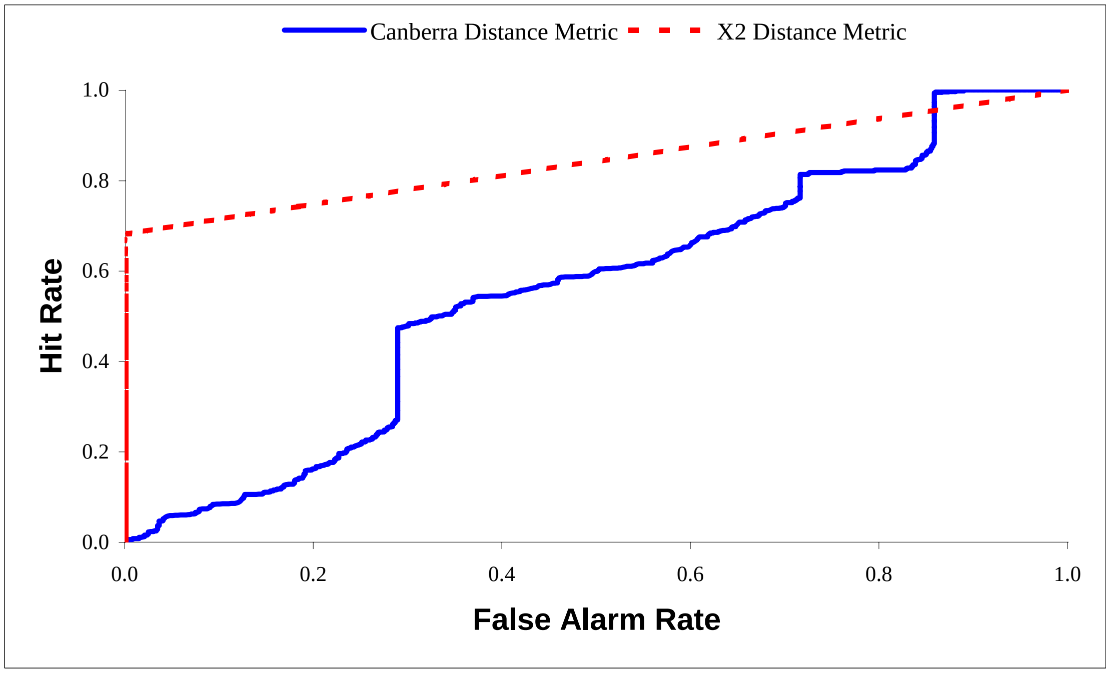


Figure 3-3. ROC curve for the third data set at the highest noise level.

Figure 3-1 shows the ROC curve for the testing results on the first small testing data set with the lowest noise level (pure normal data followed by pure intrusive data) among the three small testing data sets. Figure 3-2 shows the ROC curve for the testing results on the second small testing data set with the intermediate noise level (including 400 normal events, 530 intrusive events, 303 normal events, and then 695 intrusive events). Figure 3-3 shows the ROC curve for the testing results on the third small testing data set with the highest noise level (about 100 normal events before each intrusion session). The top-left corner of the charts in these figures represents the 100% hit rate and the 0% false alarm rate. Hence, the closer the ROC curve is to the top-left corner, the better the intrusion detection performance.

Figure 3-1 indicates that the Canberra distance metric yields a better performance than the chi-square distance metric in the condition of the lowest noise level. Figure 3-2 and Figure 3-3 indicate that the chi-square distance metric yields a better performance than the Canberra distance metric in the conditions of the intermediate noise level and the highest noise level respectively. By comparing the performance of the Canberra distance metric among the three conditions of different noise levels, we find that the performance of the Canberra distance metric drops as the noise level increases. The performance of the Canberra distance metric in the condition of the highest noise level drops almost to the diagonal line in the chart that corresponds to the expected performance of a random detector. Hence, the Canberra distance metric is sensitive to noises. On the other hand, the performance of the chi-square distance metric is consistent among different noise levels. Hence, the chi-square distance metric is robust to noises.

At all the three noise levels, the chi-square distance metric produces about the 70% hit rate at the 0% false alarm rate. Since common commands such as *vi* and *ls* in a UNIX operating system are frequently used by both normal users and intruders, we should not expect that we can signal every audit event in an intrusion session, thus having the 100% detection rate at the 0% false alarm rate.

3-3.2 Large Data Set

Since the large data set contains a large number of events, we perform the ROC analysis not based on events but based on sessions. We group the events in the large testing data set by session, count how many of the events in each session are signaled, and then compute the ratio of the number of signals to the number of events in that session as the session signal ratio. If it is an intrusion session, we expect a high session signal ratio. If it is a normal session, we expect a low session signal ratio. Figure 3-4 shows the ROC curve of the testing results on the large data set that resembles computer audit data collected from real-work information systems. A ROC curve based on session signal ratio tells us how well we distinguish the intrusion sessions from the normal sessions. The ROC curve in Figure 5 indicates that the chi-square distance metric performs better than the Canberra distance metric in distinguishing the intrusion sessions from the normal sessions based on session signal ratio.

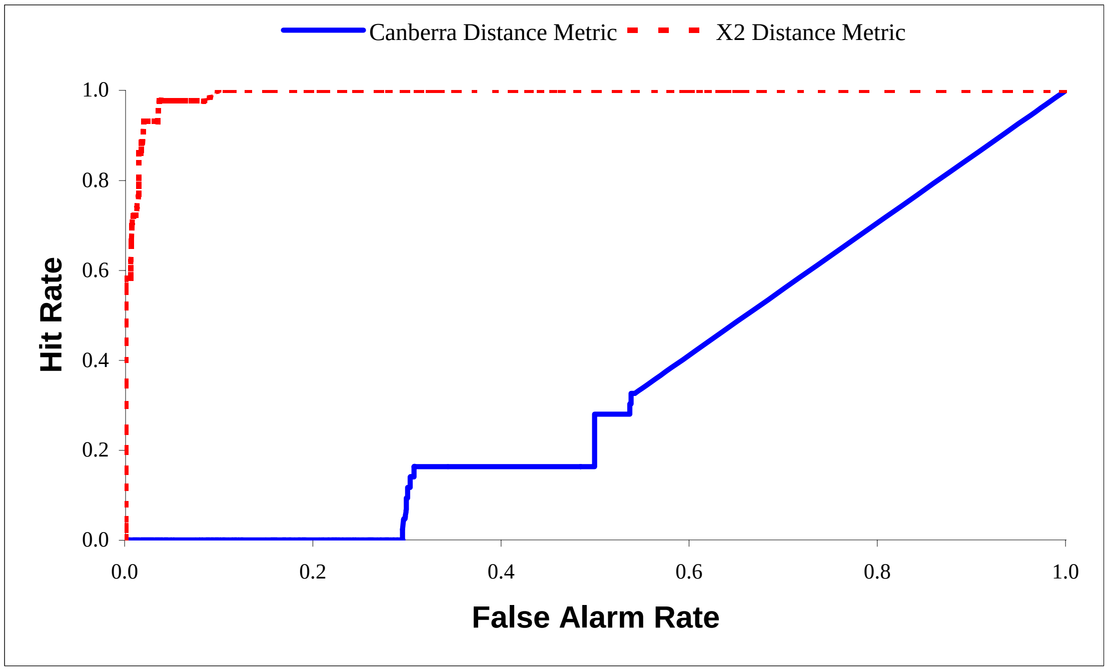


Figure 3-4. ROC curve for the large data set based on session signal ratio

3-4 Conclusion

The multivariate quality control technique based on the chi-square distance metric demonstrates consistently better performance under the conditions of different noise levels than the multivariate quality control technique based on the Canberra distance metric. The chi-square distance metric is much more robust to noises in data than the Canberra distance metric. The computation involved in the multivariate quality control technique based on the chi-square distance metric is also simple, which makes it scalable to large amounts of computer audit data for real-time intrusion detection.

Intrusions occur while normal activities are also taking place on a host machine. Even if no normal users are using the host machine at the time when an intrusion occurs, there are system programs (e.g., ftp daemon and the program monitoring the inputs from the keyboard) in the information system running all the time. Hence, computer audit data from a host machine always contain noises of normal activities. Because of the scalability and robustness of the multivariate quality control technique based on the chi-square distance metric to noises in large amounts of computer audit data, this technique is highly recommended to be included in commercial intrusion detection systems for real-time intrusion detection. Most of existing commercial intrusion detection systems use a signature recognition technique for intrusion detection. Since signature recognition techniques can detect only known intrusions, an anomaly detection technique such as the multivariate quality control technique based on the chi-square distance metric is highly desirable to complement the signature recognition technique in commercial intrusion detection systems for real-time or off-line intrusion detection.

Chapter 4: Probabilistic Networks with Undirected Links for Detecting Attacks on Information Systems

4-1 Problem Definition

This section describes our attack detection problem, including the data source, problem representation, training data, and testing data.

4-1.1 Data Source

A computer and network system within an organization typically includes a number of host machines (e.g., machines running a UNIX operation system and machines running the Windows NT operating system) and communication links connecting those host machines. Currently two sources of data have been widely used to capture activities in a computer and network system for attack detection: network traffic data and audit trail data (audit data). Network traffic data contain data packets traveling over communication links between host machines to capture activities over communication networks. Audit data capture activities occurring on a host machine. In this study, we use audit data from a UNIX-based host machine (specifically a Sun SPARC 10 workstation with the Solaris operating system), and focus on attacks on a host machine that leave trails in audit data. The Solaris operating system from the Sun Microsystems Inc. has a security extension, called the Basic Security Module (BSM). The BSM extension supports the monitoring of activities on a host by recording security-relevant events. Activities on a host machine are captured through a continuous stream of audit events.

4-1.2 Training and Testing Data

In this study, we use a sample of the audit data recording both normal activities and attack activities on host machines with Solaris 2.5. Normal activities and attack activities are simulated to produce these audit data. Normal activities are simulated according to normal activities observed in a real-world computer and network system [2]. A number of attacks are also simulated, including password guessing, use of symbolic links to gain the root privilege, attempts to gain an unauthorized remote access, etc. In the sample, the audit data of normal activities consist of 3019 audit events, and the audit data of attack activities consists of 1223 audit events. We use the first part of the audit data for normal activities as our training data set, and use the remaining audit data for normal activities and attack activities as our testing data set. The training data set consists of 1613 audit events for normal activities. The testing data set consists of 1406 audit events for normal activities and 1223 audit events for attack activities.

4-1.3 Knowledge Representation

An BSM audit record for each event contains a variety of information, including the event type, user ID, group ID, process ID, session ID, the system object accessed, and so on. Studies [20,48-49] show that types of events in information systems can be used to effectively detect many attacks. Hence, in this study we extract and use the event type from the record of each audit event. There are 284 different types of BSM audit events from Solaris 2.5 which is used to collect the audit data.

For attack detection, we want to build a long-term profile of normal activities, and to compare the activities in the recent past to the long-term norm profile for detecting a significant difference. We define activities in the recent past by opening up an observation window of size

N on the continuous stream of audit events to view the last N audit events from the current time *t*:

$E_{t-(N-1)=t-N+1}$, …, E_t, where E stands for event.

At the next time *t+1*, the observation window contains E_{t-N+2}, …, E_{t+1}. In this study, we let N equal to 100, because attack activities produce in average about 100 audit events in the data sample.

We define 284 variables (X_1, …, X_{284}) to represent 284 event types, respectively. We investigate two ways to measure the activities in the recent past and obtain values of the 284 variables from the last N audit events. One way, called the count measurement, is to count the number of a certain event type appearing in the observation window. For example, if there are 10 audit events among the N number of audit events in the observation window that fall into the 1st event type, then X_1 has the value of 10. If the 3rd event type does not show in the observation window at all, then X_3 has the value of 0. Another way, called the existence measurement, is to use 1 and 0 to represent the existence and non-existence of a certain event type in the observation window. For the same example, since the 1st event type shows up in the observation window (10 times), then X_1 has the value of 1. Since the 3rd event type does not appear in the observation window at all, then X_3 has the value of 0. Hence, for each observation window we can obtain a vector X including the values of (X_1, …, X_{284}).

4-1.4 Attack Detection Problem

Before training the norm profile, the stream of the 1613 audit events in the training data set is viewed through a moving window, which in turn creates 1514 (1613-99) window slices of audit events for the window size of 100 audit events. There is no window slice created for each of the first 99 audit events, because the current audit event and previous audit events are not sufficient to make up a complete window slice. For each window slice of audit events, a vector of (X_1, …, X_{284}) is obtained. Hence, from the training data of the 1613 audit events, we obtain 1514 vectors that we use for training a probabilistic network as the norm profile.

For the testing, 1223 audit events for attack activities and 1406 audit events for normal activities are viewed through a moving window, creating totally 2431 window slices (1124 window slices for attack activities and 1307 window slices for normal activities, numbered from No.1-1124 and No. 1125-2431). Each of the first 99 audit events for either attack activities or normal activities does not create a window slice, because the event and previous events together are not sufficient to form a complete window slice of 100 audit events.

For each window slice, a vector of (X_1, …, X_{284}) is obtained and evaluated against the norm profile to yield the probability that the normal profile supports this vector. The larger the probability is, the more support the vector receives from the normal profile, and the more likely the vector is a part of normal activities. The smaller the probability is, the less likely the vector is normal, and the more likely it is a part of attack activities.

4-2 Learning and Inference of Probabilistic Networks

The probabilistic network technique is developed based on the theory of Bayesian networks. In this section, we briefly review the theory of Bayesian networks. Then we describe the probabilistic network technique.

4-2.1 Bayesian Networks

Bayesian networks are also called Bayesian belief networks, because they are used to represent and infer the beliefs in a set of variables through probabilistic representation and reasoning [50-58]. In a Bayesian network, a set of variables make up the nodes of the Bayesian network, and a set of directed links connect pairs of nodes. A directed link from node X to node Y represents a dependency of Y on X, such as a direct influence of X on Y, a causal relationship of X and Y (X causing Y), a temporal relationship of X and Y (X preceding Y), and so on. X is called a parent of Y. Each variable has a number of states. For example, variable X may represent a particular event and has two states denoting the occurrence or non-occurrence of this event. Each node has a conditional probabilistic table that describes the probabilistic distribution of states for the corresponding variable given the states of its parent nodes. If there is no directed link to the node, the conditional probabilistic table becomes simply a probabilistic table of states for the corresponding variable. A Bayesian network is represented through a directed acyclic graph. No directed cycles are allowed in a Bayesian network.

The information in a Bayesian network provides a joint probability distribution of all the variables X_1, …, X_n in the Bayesian network. If we label these variables in an order that is consistent with their directed links in the Bayesian network,

$$\begin{aligned} & P(X_1, ..., X_n) \\ & = P(X_n \mid X_{n-1}, ..., X_1) * P(X_{n-1}, ..., X_1) \\ & = \prod_{i=1}^{n} P(X_i \mid X_{i-1}, ..., X_1) \qquad (2) \\ & = \prod_{i=1}^{n} P(X_i \mid parents(X_i)) \end{aligned}$$

Hence, the conditional probability tables in a Bayesian network provide a decomposed representation of the joint probability table for all the variables in the Bayesian network. From this joint probability table, we can derive the probability of any state involving any combination of the variables. Since many pairs of variables are conditionally independent (no directed links between them in a Bayesian network), the joint probability table is simplified into the conditional probability tables in a Bayesian network.

In many real-world problems, we are not interested in the direction of dependence but the strength of dependence. For example, in this study we are interested in how likely various audit events correlate (co-occur) in a moving window during normal activities (for a norm profile) or attack activities (for an attack profile). We may consider the causal or temporal relationship of audit events as the direction of dependence between audit events. However, since one audit event may cause or precede another audit event in some activities or vice versa in other activities, a directed cycle between these two audit events exists, which is not allowed in a Bayesian network. Therefore, in this study we consider the strength of dependence without the direction, where the dependence between audit events is characterized by the co-occurrence of audit events. We also need to develop the learning and inference algorithms for a probabilistic network with undirected links.

4-2.2 Probabilistic Networks with Undirected Links

To build a probabilistic network with undirected links, we go back to the joint probability table of the variables where no direction is implied. Then we simply make the joint probability table into a probabilistic network with undirected links. An undirected link is placed between a pair of variables if they have a strong dependency. A joint probability table is created for a group of variables which are fully dependent on each other. For example, if variables A, B, C and D are fully linked to form a fully connected graph, a joint probability table is created for the relation of variables A, B, C and D. Hence, the joint probability table of all the variables are simplified into a probabilistic network with undirected links, or in other terms, a set of smaller joint probability tables for all the nodes and relations.

4-2.3 Learning of Joint Probability Tables

The joint probability table for a node involving a single variable X is estimated from the training data as follows [54].

$$P(X = i) = \frac{N_{X = i}}{N} \tag{3}$$

where

N is the total number of observations in the training data,

$N_{X=i}$ is the number of observations with X in state *i*, and

P denotes the probability that X is in state *i*.

The joint probability table for a relation involving more than one variable such as X_1, …, X_k is estimated from the training data as follows [54].

$$P(X_1 = i_1, \ldots\ldots, X_k = i_k) = \frac{N_{X_1 = i_1, \ldots\ldots X_k = i_k}}{N} \tag{4}$$

where

N is the total number of observations in training data,

$N_{X=i}$ is the number of observations with X_1 in state i_1, …, and X_k in state i_k, and

P denotes the probability that X_1 is in state i_1, …, and X_k is in state i_k.

4-2.4 Learning of the Structure of a Probabilistic Network

The structure of a probabilistic network consists of nodes and undirected links between nodes. Given a set of variables and a number of their observations in the training data, nodes in a probabilistic network are constructed by creating one node for each variable. Undirected links between variable nodes are constructed by learning the dependence between variables from the training data.

The structure is learned through two phases. Phase I is to learn an initial structure of the probabilistic network. Phase II uses an iterative procedure of search and scoring to find the optimal structure of the probabilistic network.

To construct the initial structure of the probabilistic network in Phase I, we use the chi-square test of independence to determine the strength of dependence Q between a pair of variables X_i and X_j as follows.

$$X^2 = \sum_{k=1}^{K} \sum_{l=1}^{L} \frac{(N_{X_i = k, X_j = l} - N * P_{kl})^2}{N * P_{kl}} \tag{5}$$

$$P_{kl} = P_{k.} * P_{.l} \tag{6}$$

$$P_{k.} = P(X_i = k) = \frac{N_{X_i = k}}{N} \tag{7}$$

$$P_{.l} = P(X_j = l) = \frac{N_{X_j = l}}{N} \tag{8}$$

$$Q = [2^{\frac{d}{2}} \Gamma(\frac{d}{2})]^{-1} \int_{X^2}^{\infty} t^{\frac{d}{2}-1} e^{\frac{t}{2}} dt \tag{9}$$

$$\Gamma(x) = \int_{0}^{\infty} t^{x-1} e^{-t} dt \tag{10}$$

where

X^2 denotes the chi-square statistic,

N is the total number of observations,

$N_{Xi=k,\ Xj=l}$ is the number of observations with X_i in state k and X_j in state l,

$N_{Xi=k}$ is the number of observations with X_i in state k,

$N_{Xj=l}$ is the number of observations with X_j in state l, and

d denotes the degree of freedom which is equal to (m-1)*(n-1), m is the number of states that X_i has, and n is the number of states that X_j has.

The larger the X^2 value is, the larger difference exists between the observed frequency and the expected frequency of X_i in state *k* and X_j in state *l* (expected under the hypothesis of X_i and X_j are independent). The larger the X^2 value is, the smaller the Q value is, the less likely X_i and X_j are independent, and the stronger dependence exists between X_i and X_j.

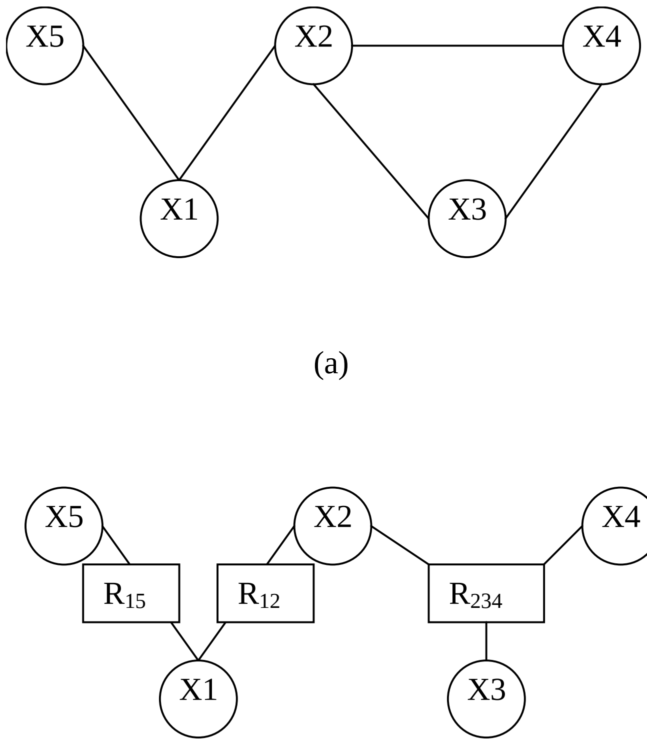
After computing the strength of dependence between every pair of the variables, we rank the pairs of variables by the ascending order of Q values. The pair of the variables that is ranked first has the strongest strength of dependence. We establish the links between the first B pairs of the variables, with a constraint that a maximum of D links can be established from a variable. The parameters B and D are introduced to control the complexity of the initial structure of the probabilistic network, so that links are established for only strongly dependent pairs of variables, while limiting the number of links from each variable. For example, considering the following rank of variable pairs with Q values.

1. (X1, X2) with Q = 0.10
2. (X2, X3) with Q = 0.15
3. (X2, X4) with Q = 0.20
4. (X3, X4) with Q = 0.25
5. (X2, X5) with Q = 0.30

6. (X5, X1) with Q = 0.35

7. (X5, X4) with Q = 0.40.

If B is set to 5 and D is set to 3, the initial structure of the probabilistic network looks like the graph in Figure 4-1. There is no link between the pair of (X2, X5), because at the maximum three links are allowed for variable X2. There is no link between the pair of (X5, X4), because the total number of links is limited up to 5.



(b)

Figure 4-1. An example of the initial structure of a probabilistic network and relations among variable nodes.

After obtaining the initial structure of the probabilistic network, we search and find all fully connected graphs that exist in the initial structure of the probabilistic network, from the largest fully connected graphs to the smallest fully connected graphs which include only two nodes. For each fully connected graph, we establish a relation node. For example, in Figure 4-1 the largest fully connected graph has three nodes, X_2, X_3, and X_4. A relation node R_{234} is established. Next, we find two fully connected graphs, each of which has only two nodes, and establish two relation nodes, R_{12} and R_{15}. Then we obtain a joint probability table for each variable node using formula (3) and a joint probability table for each relation node using formula (4). With the structure (nodes and links) and the joint probability tables, the initial probabilistic network is completed.

Phase II is based on the minimum description length principle to search for the optimal structure of the probabilistic network that best supports the training data [54]. The initial probabilistic network is evaluated to produce the following score [54].

$$Score = A * DL(TD \mid PN) + C * DL(PN) \tag{11}$$

where

TD stands for the training data,

PN stands for the probabilistic network,

DL(TD | PN) denotes the description length of the training data, given the probabilistic network, which measures the support (fitness) of the probabilistic network to the training data (the better the support, the smaller the description length),

DL(PN) represents the description length of the probabilistic network which measures the complexity of the probabilistic network,

A is a weighting factor on the measure of how well the probabilistic network supports the training data, and

C is a weighting factor on the complexity measure of the probabilistic network.

In general, the more complex the probabilistic network is, the better support (fitness) the probabilistic network can provide to the training data (including noises in the training data). The probabilistic network should closely fit the training data, while preventing the overfitting of the probabilistic network to the training data. We use a following measure of support as the description length of the training data given the probabilistic network.

$$DL(TD \mid PN) = \sum_{O_i \in TD} - \log_{10} (P(O_i \mid PN)) \qquad (12)$$

where O_i is an observation in the training data, and $P(O_i \mid PN)$ is the probability that O_i is supported by the probabilistic network. $P(O_i \mid PN)$ is determined by the equations in the next section. The smaller the DL(TD | PN) value is, the better support the probabilistic network provides to the training data.

The overfitting problem is often controlled using the Minimum Description Length (MDL) principle [54] to minimize the complexity of a model. The complexity of a probabilistic network depends on three factors: the number of variable nodes, the number of relation nodes, and the size of the joint probability table at each node. If we count the number of cells in each joint probability table, the sum of the counts over all the joint probability tables accounts for all the three factors. Hence, we let the sum of the counts be the description length (DL) of the probabilistic network.

In general, a more complex probabilistic network leads to a better support of the probabilistic network to the training data. Our goal is to look for a probabilistic network that minimizes the weighted sum of DL(TD | PN) and DL (PN) in formula (11), subject to the inherently inverse relationship between DL(TD | PN) and DL (PN).

We conduct a heuristic search for an alternative probabilistic network with a smaller score (a better probabilistic network) than the initial probabilistic network. The only way to change the initial probabilistic network is to change its links, since the number of variable nodes in the probabilistic network is fixed and the joint probability tables are determined from the training data. In the heuristic search, we explore the one-link change for all possible pairs of variable nodes in the initial probabilistic network. If there is a link between a pair of variable nodes, a one-link change for the pair of variable nodes is to remove the link. If there is no link between the pair of variable nodes, a one-link change is to add a link between the pair of variable nodes. For a probabilistic network with *n* nodes, there are n*(n-1)/2 possible pairs and thus the same number of possible one-link change. Each one-link change leads to a new structure of the probabilistic network. For each new structure, we compute the joint probability tables and the evaluation score in formula (11). We then compare the evaluation scores for the n*(n-1)/2 probabilistic networks, and select one with the smallest evaluation score. Then the next round of search and scoring continues with the selected probabilistic network, until the evaluation score no longer improves (reaching a global or local minimum).

4-2.5 Inference Algorithm

The support of the probabilistic network to an observation O_i in the training data in formula (12), P(Oi | PN), is determined as follows. The observation O_i is a vector of (X_1, …, X_n).

$$P(X_1,...,X_n) = P(X_1) * P(X_2 \mid X_1) * ... * P(X_n \mid X_{n-1},...,X_1) \qquad (13)$$

The conditional probability $P(X_k \mid X_{k-1}, \ldots, X_1)$ represents the updated joint probability table of X_k based on new evidences on $X_{k-1}, \ldots, X_1$, where k = 2, …, n. Each $P(X_k \mid X_{k-1}, \ldots, X_1)$ term in formula (13) is determined through probability update and propagation as shown below.

If Xi and Xj are linked through relation R, P(Xj | Xi) is determined by the following formulas which are similar to those for probability updating in a Bayesian network [50].

$$P^{(m)}(R) = normalize\left[P^{(m-1)}(R) \frac{P^{(k)}(X_i)}{P^{(0)}(X_i)} \right] \qquad (14)$$

$$P^{(n)}(X_j) = normalize\left[P^{(n-1)}(X_j)\,\frac{m\arg inalize\left[P^{(m)}(R)\right]}{P^{(0)}(X_j)} \right] \qquad (15)$$

where

R denotes a relation,

X_j and X_i are variable nodes involved in relation R,

$P^{(m)}(R)$ denotes the updated joint probability table of relation R based on the new evidence on X_i,

$P^{(m-1)}(R)$ denotes the updated joint probability table of relation R based on another new evidence which is also involved in R,

$P^{(0)}(R)$ is the prior state distribution of R before any update,

$P^{(n)}(X_j)$ denotes the updated joint probability table of X_j based on the updated joint probability table $P^{(m)}(R)$ if there is any update of R, or based on the prior joint probability table $P^{(0)}(R)$ if there is no update of R,

$P^{(n-1)}(X_j)$ denotes the updated joint probability table of X_j based on another relation which also involves $X_{j,}$ and

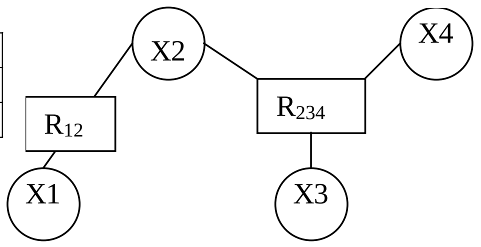
$P^{(0)}(X_j)$ is the prior state distribution of X_j before any update.

Formula (14) is used to update the joint probability table of relation R from a new evidence on X_i. Formula (15) is used to update the joint probability table of X_j from the updated joint probability table of relation R.

Figure 4-2 shows a probability network consisting of four variable nodes (X_1, X_2, X_3, and X_4) and two relation nodes (R_{12} and R_{234}). The initial joint probability tables are given in part (a) of Figure 4-2. Part (b) of Figure 4-2 shows the computations for the probability update and propagation. Given a new evidence on X_1, the updated joint probability table of X_2 is determined by first updating the joint probability table of relation R_{12} using formula (14).

$P(X_2) = \{y=0.2, n=0.8\}$

$P(X_4) = \{y=0.1, n=0.9\}$



$P(R_{12})$ =

	X_2 = y	X_2 = n
X_1 = y	0.2	0.16
X_2 = n	0	0.64

$P(X_1) = \{y=0.36, n=0.64\}$
$P^*(X_1) = \{y=1, n=0\}$

$P(X_3) = \{y=0.27, n=0.73\}$
$P^*(X_3) = \{y=1, n=0\}$

$P(R_{234})$ =

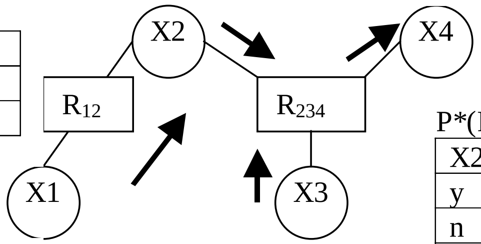
X2	X4	X3	$P(R_{234})$
y	y	y	0.02
n	y	y	0.07
y	n	y	0.18
n	n	y	0
y	y	n	0
n	y	n	0.01
y	n	n	0
n	n	n	0.72

*: indicates an evidence.

(a)

$P^*(X_2) = \{y=0.56, n=0.44\}$

$P^*(X_4) = \{y=0.16, n=0.84\}$



$P^*(R_{12})$ =

	X_2 = y	X_2 = n
X_1 = y	0.56	0.44
X_2 = n	0	0

$P^*(X_1) = \{y=1, n=0\}$

$P^*(X_3) = \{y=1, n=0\}$

$P^*(R_{234})$ =

X2	X4	X3	$P^*(R_{234})$
y	y	y	0.094
n	y	y	0.064
y	n	y	0.842
n	n	y	0
y	y	n	0
n	y	n	0
y	n	n	0
n	n	n	0

$\rightarrow$ indicates the direction of probability update and propagation.

(b)

Figure 4-2. Probability update and propagation in a probabilistic network

$$P(X_1 = y, X_2 = y) = 0.2 * \frac{1}{0.36} = 0.56$$
$$P(X_1 = y, X_2 = n) = 0.16 * \frac{1}{0.36} = 0.44$$
$$P(X_1 = n, X_2 = y) = 0 * \frac{0}{0.64} = 0$$
$$P(X_1 = n, X_2 = n) = 0.64 * \frac{0}{0.64} = 0$$

Since the sum of the above four values is 1, the normalization is not needed. We then update the joint probability table of X2 by marginalizing X1 out of the updated joint probability table of relation R12 as follows.

$$P(X_2 = y) = 0.56 + 0 = 0.56$$
$$P(X_2 = n) = 0.44 + 0 = 0.44$$

For another example, $P(X_4 \mid X_3, X_1)$ requires the updated joint probability table of X4 given the new evidences on X_1 and X_3. To determine $P(X_4 \mid X_3, X_1)$, we update the joint probability table of R_{234} using the updated evidence on X_2. The updated joint probability table of R_{234} is again updated using the new evidence on X_3. By marginalizing X_2 and X_3 out of the twice-updated joint probability table of R_{234}, we obtain the updated joint probability table of X_4 as shown in part (b) of Figure 4-2.

4-3 Probabilistic Networks for Cyber Attack Detection

As discussed in section 4-1.3, we obtain two sets of (X_1, …, X_{284}) vectors from the training data set: one set for the count measurement, and another set for the existence measurement. The learning and inference algorithms of the probabilistic network apply to variables with a finite number of discrete values as states. In the set of the (X_1, …, X_{284}) vectors with the existence measurement, each variable has two states: existence (with the value of 1) and non-existence (with the value of 0). In the set of the (X_1, …, X_{284}) vectors with the count measurement, each variable takes a count number. Although a count is not exactly a continuous value, it is a discrete value with possibly no upper limit. For our attack detection problem, the upper limit of a count is equal to the size of the moving window. That is, the largest count for a certain type of audit event is equal to the total number of audit events in the moving window, when all audit events in the moving window are the same type. If we take each possible value of the count as one state of a variable, the variable may end up with possibly too many states. Since many of such states are just slightly different and do not always appear, it would be a waste of computer resources if we take each possible count as one state. Such a waste of computer resources is not desirable, especially when we deal with a large-scale problem.

Therefore, we want to transform each set of (X_1, …, X_{284}) vectors with the count measurement so that variables take a reasonable number of discrete states. A variety of methods exist to discretize continuous values [54-58]. Those methods generally fall in two categories. One category of the methods determine a set of dividing points to yield the discrete segments of a continuous variable by evaluating whether this set of dividing points lead to a better fitted model to the training data. Another category of the methods determine a set of dividing points using a fixed formula, leading to either a linear segmentation or non-linear segmentation. In this study, we use the following non-linear segmentation formula to simplify the computation in learning:

$$\begin{aligned} State &= 1 + \log_2 Count \quad && if\ Count > 0 \qquad (16) \\ &= 0 && \text{if Count} = 0 \end{aligned}$$

The two kinds of measurements (count and existence) produce two different sets of (X_1, …, X_{284}) vectors to train two different probabilistic networks, respectively. Each probabilistic network had 284 variable nodes. The parameter B for the maximum number of links in a probabilistic network was arbitrarily set to 70. The parameter D for the maximum number of links from a variable node was arbitrarily set to 3. The weighting factor for the entropy of the training data in a probabilistic network was arbitrarily set to 9. The weighting factor for the description length of a probabilistic network was arbitrarily set to 1.

The testing data contain the audit data for both attack activities and normal activities. When a vector of (X_1, …, X_{284}) from a window slice of the testing data is presented to a probabilistic network trained with the audit data of normal activities (the norm-based probabilistic network), the probability that this vector is supported by the norm-based probabilistic network is determined by formula (13). The larger the probability is, the more likely activities in the window slice are normal.

It is possible that a vector of (X_1, …, X_{284}) from the testing data presents a state of a variable and/or a state of a relation among several variables which are not encountered during the training and are thus not covered by the joint probability tables of the trained probabilistic network. While using formula (13) to infer the support probability of the trained probabilistic network to this vector of (X_1, …, X_{284}), we assign a state, which is not available in the trained probabilistic network, a probability of 0.00001 which is close to zero and is much smaller than any existing probabilities in the joint probability tables of the trained probabilistic network.

We do not assign the probability of zero to such states, because the probability of zero would make the final result from formula (13) become zero. This would make it impossible to distinguish normal activities with noises from attack activities, since noises in normal activities may introduce new states. The amount of new states from noises in normal activities is expected much less than the amount of new states from attack activities. By assigning a small probability rather than zero to the new states, the smaller effect of noises in normal activities on the final result of formula (13) becomes distinguishable from the larger effect of attack activities on the final result of formula (13).

4-4 Results and Discussions

During the testing, we compute the probabilities that the two trained probabilistic networks support the testing data. Table 4-1 summarizes the statistics of the testing results. For each probability network, we compute the minimum, maximum, average and standard deviation of the probabilities that are produced for each kind of the testing data (the testing data of normal activities – normal data, and the testing data of attack activities – attack data).

Table 4-1. The statistics of the testing results.

Training	Measurement	Testing	Minimum	Maximum	Average	Standard Deviation
Normal data	Existence	Normal data	4.89E-05	2.64E-01	1.29E-01	1.08E-01
Normal data	Existence	Attack data	3.46E-113	1.48E-21	7.90E-24	1.08E-22
Normal data	Count	Normal data	6.04E-18	1.26E-02	1.96E-03	2.24E-03
Normal data	Count	Attack data	4.28E-127	1.09E-29	8.74E-32	8.44E-31
Attack data	Existence	Normal data	9.51E-47	3.00E-33	5.28E-34	1.14E-33
Attack data	Existence	Attack data	4.84E-82	4.78E-05	1.01E-07	2.02E-06
Attack data	Count	Normal data	8.14E-46	7.13E-26	2.83E-28	4.03E-27
Attack data	Count	Attack data	1.56E-82	6.60E-08	4.80E-10	4.72E-09

The two norm-based probabilistic networks, which are trained with the normal data (audit data of normal activities) for two kinds of measurements respectively, produce much larger probability values for the normal data than the attack data during the testing. The probabilities for event numbers 1125-2431 (the normal data) are much larger than the probabilities for event numbers 1-1124 (the attack data). A larger probability means more support of the norm-based probabilistic networks to the data. For the existence measurement, there exists a huge gap between the minimum probability for the normal testing data (4.89E-05) and the maximum probability for the attack testing data (1.48E-21). For the count measurement, there also exists a huge gap between the minimum probability for the normal testing data (6.04E-18) and the maximum probability for the attack testing data (1.09E-29).

These results indicate that for both kinds of measurement we are able to clearly distinguish the normal activities from the attack activities during the testing, using any probability value in the gaps as the decision threshold. If the probability of the testing data from a moving window is greater than the decision threshold, the activities in the moving window are classified as normal. Otherwise, the activities in the moving window are classified as attack.

In overall, the probability for the count measurement on the testing data within a moving window is smaller than the probability for the existence measurement on the same testing data for two reasons. First, we use the small probability value of 0.00001 for a new state that appeared in the testing but did not show in the training, when we use formula (13) to calculate the final probability of a vector from the moving window. Second, the count measurement creates more of such new states.

In summary, the norm-based probabilistic networks demonstrate promising performance in detecting attack activities during the testing, regardless of which measurement method is used. The results from this study encourage the further investigation of the probabilistic network technique and its application to attack detection.

Note that the results of this study are obtained using an arbitrary set of parameters for the probabilistic networks, such as parameter B for the maximum number of links in a probabilistic network, parameter D for the maximum number of links from a variable node, the weighting factor for the entropy of the training data in a probabilistic network, and the weighting factor for the description length of a probabilistic network. In general, the larger parameters B and D are, the better a probabilistic network can fit the training data. However, a better fit does not necessarily lead to a better testing performance due to the over-fitting problem. Just as many training parameters in an artificial neural network must be determined empirically for a particular application [54], these parameters for a probabilistic network can be determined empirically using some training and testing data.

Chapter 5: Decision Tree Classifiers for Computer Intrusion Detection

5-1 Decision Tree Technique

There are two types of variables used in the decision tree technique [40]: the target variable or class, and the predictor variable(s). During training, the decision tree technique partitions the data examples labeled by the target class recursively until a stopping criterion is met. After each partition, the set of training examples falling in a branch of the decision tree has less inconsistency with respect to the target class. A typical stopping criterion for not further partitioning a branch is that all the examples should be of the same class or target value, which then produces a leaf in the decision tree.

The variables used in the decision tree can be continuous or nominal. If the predictor variables are continuous, some methods exist to automatically transform them into categorical variables. For a continuous target variable, some algorithms such as GINI use methods such as Least-Squared Deviation (LSD) to find the split of the continuous values and transform the target variable into a categorical variable.

A decision tree can be used as a classifier or to reveal the underlying patterns of data [59-61]. As a classifier, a decision tree is used to divide a problem region into several sub-regions with all the examples (data points) in one sub-region having the same target value. The sub-regions can then be used to classify the target value of new cases. The structure of decision tree also shows the relationship between the predictor variables and the target variable for a given problem. This study uses a decision tree as a classifier, called a decision tree classifier (DTC).

In this study, we use a commercial software package from SPCC Inc., AnswerTree 2.0, to develop our decision tree classifiers. We use two of the four decision tree algorithms provided in AnswerTree 2.0: CHAID and GINI [62]. CHAID uses a chi-square test to find the most significant split points on the predictor variables. GINI is essentially CART [59-61] with the GINI index. The GINI impurity index is defined as the possibility at a test node that a randomly chosen case is classified incorrectly. We choose CHAID because it can produce non-binary trees. GINI produces only binary trees.

5-2 Problem Definition

5-2.1 Functions of decision tree classifiers in signature recognition

The application of DTC to intrusion detection has two objectives [63-64]. First, the decision tree technique is used to automatically learn the signatures of normal activities and intrusive activities. Such signatures are stored in the tree structure inducted from the historic data of activities in a computer network system. A branch from the root of a decision tree to a leaf of a decision tree corresponds to a signature pattern. The lead contains the training examples falling into this lead. We can assign the IW value to each leaf based on the classes of the training examples contained in each leaf. In addition, each leaf can be considered to represent a general state of activities in a computer network system. That is, the decision tree classifies the observed activities in the computer network system into a state of activities.

5-2.2 Training and testing data sets

The Basic Security Module (BSM) in the Solaris operating system can be used to audit activities in a host machine with the Solaris operating system. We use BSM audit data as the training and testing data in this study. The audit data consist of audit events in sequence. Each audit record for an audit event contains several attributes. Some examples of the attributes are the event type, user ID, process ID, command, time, and remote IP address that are associated with an audit event.

In our study, we use only the information of the event type, because such information has led to good detection performance in many studies. Hence, for each event, we keep only the event type attribute. There are 284 different event types in the Solaris operating system. However, only 30 different event types appear in our training data set. So there are 30 possible categorical values for the event type variable. There is a class label for each event indicating whether the event is normal or intrusive. The event is labeled as 1 if it is intrusive; otherwise, it is labeled as 0. The class label is the target variable. The target value is provided for each event in the training data set.

We also have the class label of each event in the testing data set, so that we can compare the true class label of each event in the testing data with the class label obtained from DTC to evaluate the detection performance of DTC. If a truly intrusive event in the testing data set is classified as intrusive by DTC, it is a hit. If a truly normal event in the testing data set is classified as intrusive by DTC, it is a false alarm. A better DTC should produce more hits and fewer false alarms.

BSM audit data of normal activities come from the MIT Lincoln Laboratory by simulating normal activities on a host machine with the Solaris operating system. BSM audit data of intrusive activities are also obtained by simulating various intrusion scenarios. The audit data of normal activities contain 3019 audit events. The audit data of intrusive activities contain 1751 audit events. The 1613 normal events in the training data come from the first part of the 3019 normal events. The 528 intrusive events in the training data set come from the first part of the 1751 intrusive events. Hence, the training data set contains 1613 normal events and 526 intrusive events. The testing data set contains 1406 normal events and 1225 intrusive events.

5-2.3 Problem representation

The class label of an event provides the value of the target variable for this event. The predictor variables use only the information of the event type. The decision on these predictor variables is critical, and impacts the detection performance of DTC.

One simple method is to use the event type as the only predictor variable, leading to a "single event" DTC in this study. For each event in an event stream, we obtain a value of the predictor variable for this event. Considering an event stream as time series data [65], we also develop three other methods of feature selection: "moving window", "EWMA vector", and "state ID".

In the "moving window" method, we use an observation window to extract the attributes of the event sequence in this observation window. As shown in Figure 5-1, we move an observation window along an event stream. A snapshot through the observation window at a given time gives a segment of the event stream – an event sequence. The event sequence in the observation window of size 4 shown in Figure 5-1 is Event Type 7, Event Type 6, Event Type 3, and Event Type 4. There are 30 variables for the 30 event types in our data respectively, (X1, …, X30). Each variable represents the existence (1 for existence and 0 for non-existence) of the corresponding event type in the observation window. Hence, for each event in the training and testing data, we obtain an observation vector of (X1, …, X30). After this window slides across the entire event stream, a new set of data vectors is produced. If the window size is greater than two units of the event stream, the new data unit contains the temporal information. In this study, the window size is 10. A decision tree based on this method is called the "moving window" classifier.

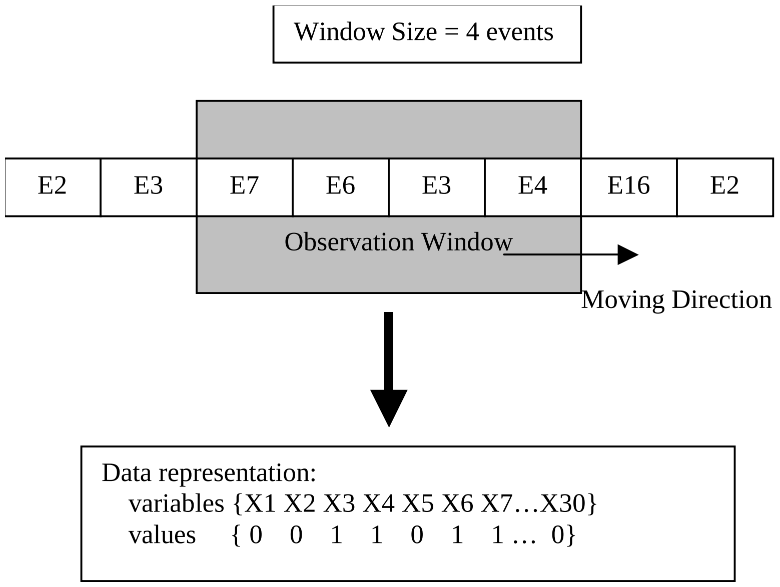


Figure 5-1. The use of a moving window to obtain a representation of the event stream.

In the "EWMA vector" method, we again use 30 variables for 30 event types respectively, (X1, …, X30). These 30 variables are the predictor variables. However, each variable represents the smoothed value of the occurrence frequency of the corresponding event type in the recent past from the current event. That is, (X1, …, X30) captures the frequency distribution of the 30 event types in the recent past of the event sequence from the current event. The smoothing is based on the Exponentially Weighted Moving Average (EWMA) technique. For the current event *n*, we calculate X(n) using the following formula:

$X_i(n) = \lambda * 1 + (1-\lambda) * X_i(n-1)$ if the current event *n* belongs to the ith event type, (1)

$X_i(n) = \lambda * 0 + (1-\lambda) * X_i(n-1)$ if the current event *n* is not the ith event type,

where $X_i(n)$ is the smoothed observation value of the *i*th variable for the current event, and λ is a smoothing constant which determines the decay rate. The larger the λ value, the larger the decay rate. According to formula (1), the occurrence of a certain event type in the current event has the weight of λ in computing the frequency of this event type in the recent past from the current event, the first event before the current event has the weight of λ*(1-λ), the second event before the current event has the weight of $\lambda*(1-\lambda)^2$, and so on. Hence, the further away an event is from the current event, less weight the event receives in computing the frequency distribution of the event types. With this "EWMA vector" approach, we add the aging into the observation value of (X1, …, X30). In this study, we initialize $X_i(0)$ to be 0 for i = 1,…,30. We let λ be 0.3 – a typical value for the smoothing constant [66], which is approximately equivalent to a window size of about 10 events since the weight for the 9th event from the current event drops to almost zero. Hence, for each event in the training and testing data, we obtain an EWMA vector of (X1,…, X30). This representation method leads to a decision tree, called the "EWMA vector" classifier.

In the "state-ID" method, we first use the "single event" classifier to classify the single event in an event stream into a state as discussed previously. Using the state information, we transform the original event stream into a state stream. Then we apply the "moving window" classifier to the state stream to produce the existence-based "state-ID" classifier. The size of the moving window is 10. In addition, we also produce a count-based "state-ID" classifier for which X_i represents the frequency count (not existence) of the ith event type in the observation window.

5-2.4 Impact of window size

In the above sections, we show the design of DTC based on different feature selection methods. Another interesting issue is the impact of window size and noises in data on the detection performance of those DTC. In the "moving window" method and the "state-ID" method, we can set different window sizes. In addition to the window size of 10, we also test the window size of 100 to investigate the impact of the window size on the detection performance of DTC. For the "EWMA vector" method, the λ value equivalent to the window size of 100 is 0.04.

5-2.5 Impact of noises in data

Our training and testing data sets contain "pure" data in that each data set has all the normal events together at first and then all the intrusive events at last. However, in reality, intrusive activities usually occur in a computer network system while there are also normal activities as background noises in the computer network system. Hence, our audit data from a host machine have a mixture of normal activities with intrusive activities.

We use the original training data set and the original testing data to create two new noisy data sets. For each data set, the normal events and intrusive events in the "pure" data set are randomly mixed while maintaining the sequential order of all the normal events and the sequential order of all the intrusive events.

Obviously the detection performance of the "single event" classifier on the pure data sets and the noisy data sets should remain the same, as this classifier looks at only a single event at a time. For the "moving window" classifier and the "state-ID" classifier, it is difficult to determine the class label of an event sequence with the mixture of normal and intrusive events. For the "EWMA vector" method, the class label is based on the class label of the current event. Hence, we investigate the impact of the noises in data using only the "EWMA vector" classifier.

We call the “EWMA vector” classifier based on the original data sets as the “pure” classifier. For noisy data sets, we produce two “EWMA vector” classifiers. One classifier is called the “sorted” classifier, because the classifier is built using the original “pure” training data set and is tested using the noisy testing data set. The training of DTC with “pure” data is possible, because we have the class label for each event which can be used to sort the noisy data into “pure” data. Another classifier for the noisy data sets is called the “noisy” classifier that is built using the noisy training data set and is tested using the noisy testing data set. Table 5-1 shows the difference of the three classifiers.

Table 5-1. "Pure", "sorted" and "noisy” classifiers

	“pure” classifier	“sorted” classifier	“noisy” classifier
Training data set	original, pure	original, pure	Noisy
Testing data set	original, pure	noisy	noisy

5-3 Results and Discussions

5-3.1 Performance analysis

After a DTC is produced, we compute the IW value for each leaf in the DTC by averaging the target value of all the training examples falling into this leaf. During testing, a data point is passed from the root of the DTC to a leaf. Thus this data point gets its predicted IW value for the IW value of this leaf.

One way to carry out the performance analysis is to calculate the mean, maximum and minimum of predicted IW values for the group of the normal events in the testing data set and for the group of intrusive events in the testing data set respectively. This kind of performance analysis provides the descriptive statistics of the detection performance. It can provide some descriptive information about what IW values we obtain for the normal events in the testing data set, what IW values we obtain for the intrusive events in the testing data set, and how these two groups of IW values are separated from each other.

Another way to carry out the performance analysis is through the Receiver Operating Characteristic (ROC) analysis. Given the predicted IW values and true IW values of the events in the testing data, we can set a signal threshold and use the signal threshold to determine whether DTC signals an event in the testing data based on its predicted IW value. If the predicted IW value of an event is greater than the signal threshold, an alarm signal is produced for this event; otherwise, no signal is produced for this event. By comparing the signals on the events in the testing data with the true class labels of these events, we obtain a hit rate which is the ratio of the number of signals on the intrusive events in the testing data set to the total number of intrusive events. We also obtain a false alarm rate, which is the ratio of the number of signals on the normal events in the testing data set to the total number of the normal events. By varying the value of the signal thresholds, we obtain pairs, (hit rate, false alarm rate) which are plotted as an ROC curve. The closer the ROC curve to the top-left corner (representing the 100% hit rate at the 0% false alarm rate) of the ROC chart, the better the detection performance.

5-3.2 Performance of the "single event" classifier

Only the CHAID algorithm in AnswerTree produces a DTC using the training data. The GINI algorithm fails to produce a DTC using the training data. Table 5-2 presents the descriptive statistics of the predicted IW values for the testing data set. We notice only a small difference in the mean IW value between the normal events and the intrusive events.

Table 5-2. Statistics for the “single event” classifier (CHAID)

IW Value	Min	Max	Mean	Standard Deviation
Normal	0.00	0.4615	0.2001	0.113
Intrus.	0.00	1.00	0.368	0.255

Figure 5-2 shows the ROC curve for the “single event classifier”. This curve is a little bit above the diagonal of the ROC chart. Although at some signal thresholds the hit rates are high, the false alarm rates at those signal thresholds are also high.

5-3.3 Performance of “moving window” classifiers

The tree structures produced from the CHAID and GINI algorithms are the same. Table 5-3 shows the statistics for the normal events and intrusive events in the testing data set. It is obvious that the predicted IW values differ considerably between the normal events and the intrusive events.

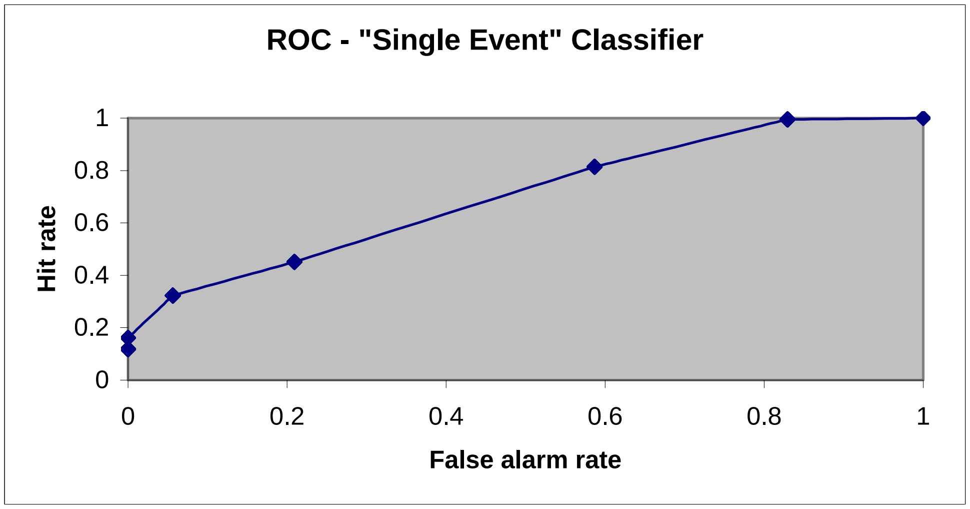


Figure 5-2. ROC analysis for the “single event” classifier based on CHAID algorithm.

Table 5-3. Statistics for the "moving window" classifier (CHAID/GINI).

IW Value	Min	Max	Mean	Standard Deviation
Normal	0.00	0.0887	0.0562	0.0428
Intrus.	0.0887	1.00	0.7902	0.3838

Figure 5-3 shows the ROC curve of the "moving window" classifier. The detection performance of the "moving window" classifiers is much better than the "single event" classifier. For a wide range of signal thresholds, the hit rate remains at about 77% while the false alarm rate is 0%.

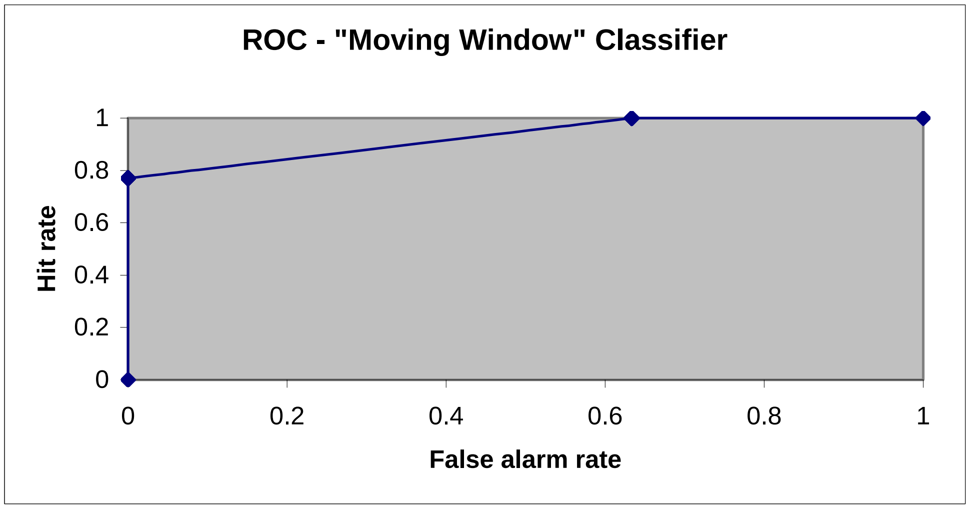


Figure 5-3. ROC analysis for moving window classifier (CHAID-GINI).

5-3.4 Performance of "EWMA vector" classifiers

The "EWMA vector" classifiers using the CHAID and GINI algorithms produce similar results. Table 5-4 gives the statistics for the classifier based on the CHAID algorithm. The normal events and the intrusive events in the testing data are clearly separated by their predicted IW values. The detection performance of this classifier is even better than that of the “moving window” classifier.

Table 5-4. Statistics for EWMA vector classifier (CHAID).

IW Value	Min	Max	Mean	Standard Deviation
Normal	0.00	0.00	0.00	0.00
Intrus.	0.00	1.00	0.881	0.324

From the ROC curves of the “EWMA vector” classifiers as shown in Figure 5-4, it appears that the “EWMA vector” classifiers produce the best detection performance among all the DTC. For the GINI algorithm, for example, the hit rate remains at about 87% while the false alarm rate is 0% for most signal thresholds.

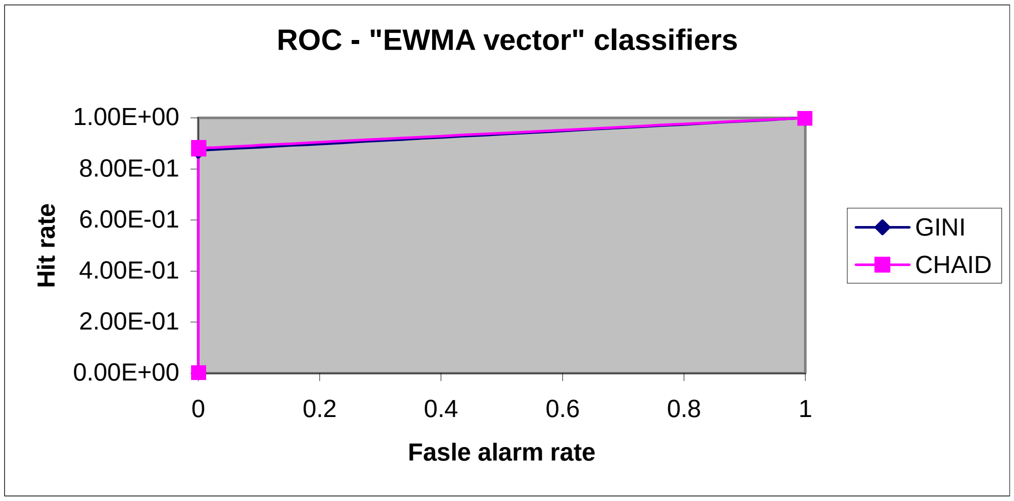


Figure 5-4. ROC analysis for EWMA vector classifiers (CHAID and GINI).

5-3.5 Performance of “State ID” classifiers

Since we use the GINI and CHAID algorithms to build our classifiers, there are a total of four classifiers produced from the state ID data. The statistics in Table 5-5 show that the difference in the predicted IW values between the normal events and the intrusive events are very large, comparable to that from the “moving window” classifier. The standard deviations are small here. This implies that we can detect the normal events and the intrusive events accurately. The count-based classifiers provide better detection performance than the existence-based classifiers.

Figure 5-5 and Figure 5-6 show the ROC curves for the two existence-based “state ID” classifiers and the two count-based “state ID” classifiers respectively. For a range of signal thresholds, the hit rates of these classifiers all reach around 96% while the false alarm rates stay at about 6.6%.

Table 5-5 Statistics for state-ID classifiers.

Classifiers		Mean/ Normal	Mean/ Attack	S.D./ Normal	S.D./ Attack
“Existence”	CHAID	0.0655	0.8611	0.2192	0.2240
	GINI	0.0593	0.8410	0.2100	0.2939
“Count”	CHAID	0.0408	0.9217	0.1956	0.2661
	GINI	0.0426	0.9170	0.1983	0.2755

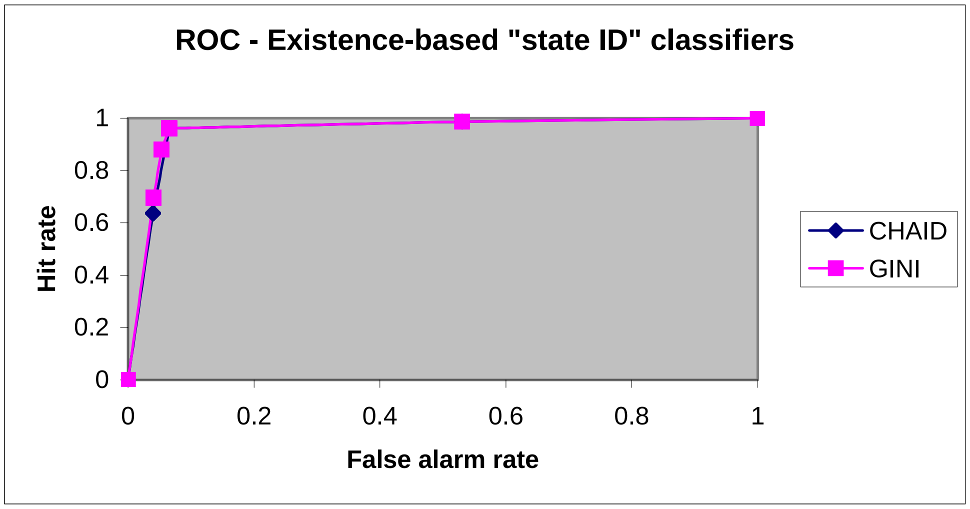


Figure 5-5. ROC curves for the existence-based "state ID" classifiers.

Therefore, the state information generalization from the "single event" classifier improves the detection performance of the "moving window" classifier.

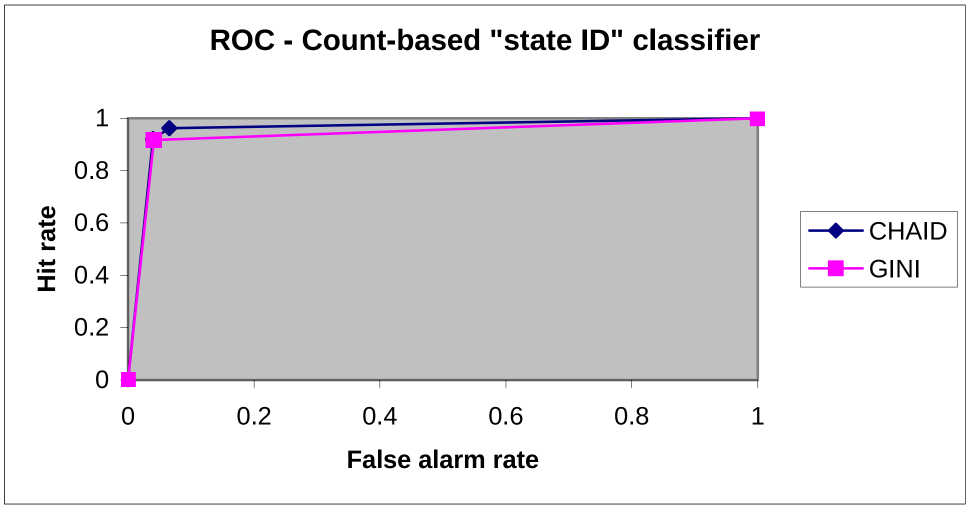


Figure 5-6. ROC curves for the count-based "state ID" classifiers.

5-3.6 Performance of classifiers under a different observation window size and a different smoothing factor λ

The above testing results are for the λ value of 0.3 and the equivalent window size of 10. Figure 5-7 shows the ROC curve of the "EWMA vector" classifier for the λ value of 0.04 based on the CHAID algorithm in comparison with the "EWMA vector" classifier for the λ value of 03.

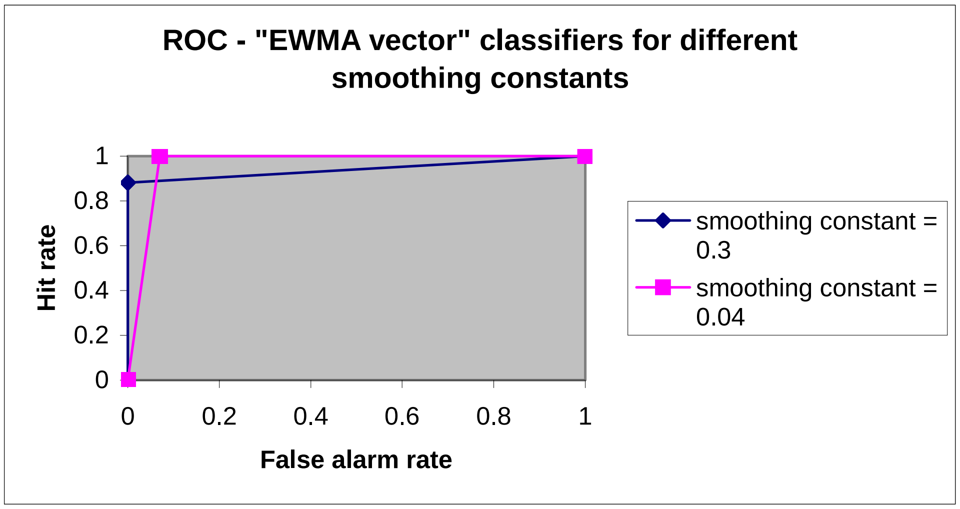


Figure 5-7. ROC curves for the "EWMA vector" classifiers with different smoothing constants based on the CHAID algorithm.

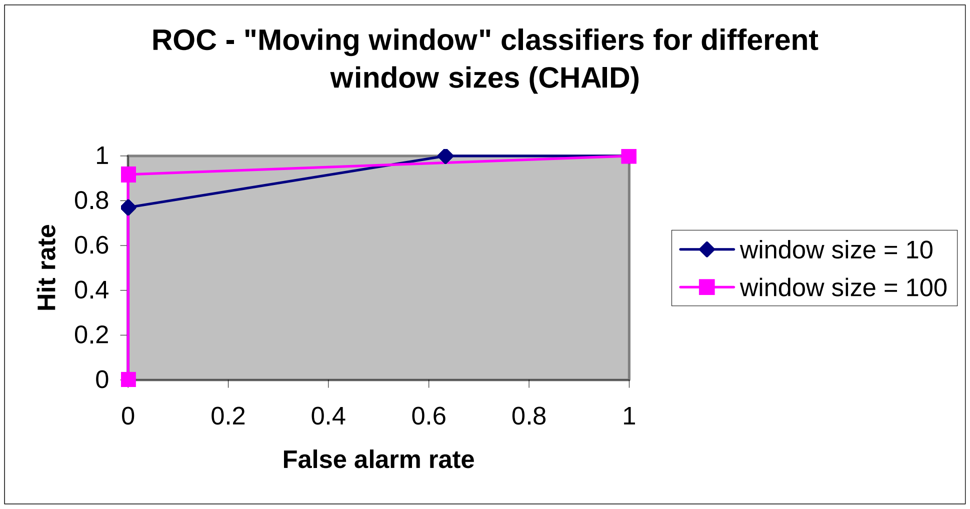


Figure 5-8. ROC curves of the "moving window" classifiers for different window sizes based on the CHAID algorithm.

There is a significant performance difference between these two ROC curves in Figure 5-7. If we let the false alarm rate be less than 10%, the hit rate of the classifier with the smooth constant of 0.04 is better than that of the classifier for the smooth constant of 0.3. This implies that for these data sets a longer observation window produces better detection performance.

From the ROC curves of the "moving window" classifiers for different moving window sizes in Figure 5-8, it appears that the longer window size produces better detection performance for these data sets, and similar performance to that of the "EWMA vector" classifiers.

5-3.7 Performance of "pure", "sorted" and "noisy" classifiers

The ROC curves of these three classifiers are shown in Figure 5-9 and Figure 5-10. It appears that the detection performance of "sorted" and "noisy" classifiers degrades sharply in comparison with that of the "pure" classifier. In the "EWMA vector" method, the observation value of each variable for the current event depends on not only the current event but also the events before the current events in the recent past. Hence, noisy data reduce the difference in the observation value between the normal event and the intrusive event, thus degrading the detection performance.

From the ROC curves in Figure 5-9 and Figure 5-10, it appears that the detection performance of the “noisy” classifier seems a little better than that of the “sorted” classifier. This implies that when the noise level of the testing data is similar to the noise level of the training data, better detection performance may be produced. Another finding is that different DTC algorithms have different impacts on the detection performance of the “noisy” classifier. This implies that some DTC algorithms may be more robust to noise than the others.

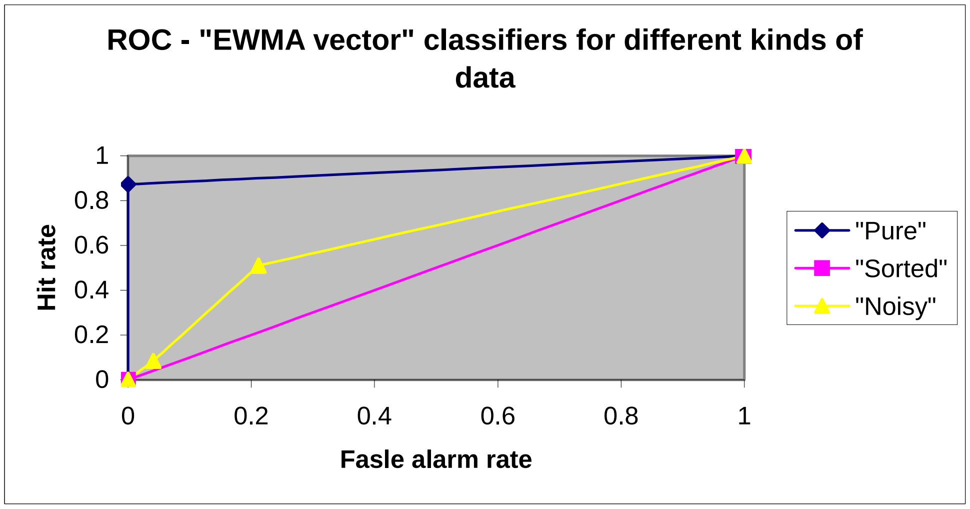


Figure 5-9. ROC curves of the “EWMA vector” classifiers based on the GINI algorithm for three kinds of data.

5-4 Summary

In this paper we show different methods of designing decision trees for automatic signature recognition in intrusion detection. Several different types of decision tree classifiers based on the event type information of audit event data have been designed and tested. We compare their detection performance. From the analysis and comparison of their detection performances, decision tree classifiers seem to show a great potential for signature recognition.

This study provides some insights into the application of the decision tree technique and possibly other data mining techniques to intrusion detection. First of all, different ways of selecting data features of data appear to have large impact on the detection performance. Secondly, parameters such as the window size and noises in data can also affect the detection performance of DTC.

There are still many issues that need to be investigated, including the scalability of decision tree algorithms for large data sets and the ability of decision tree algorithms to perform the incremental learning, because new data of normal and intrusive activities may become available over time. Since the amount of audit data from a computer network system can be huge, DTC must be computationally feasible for large data sets.

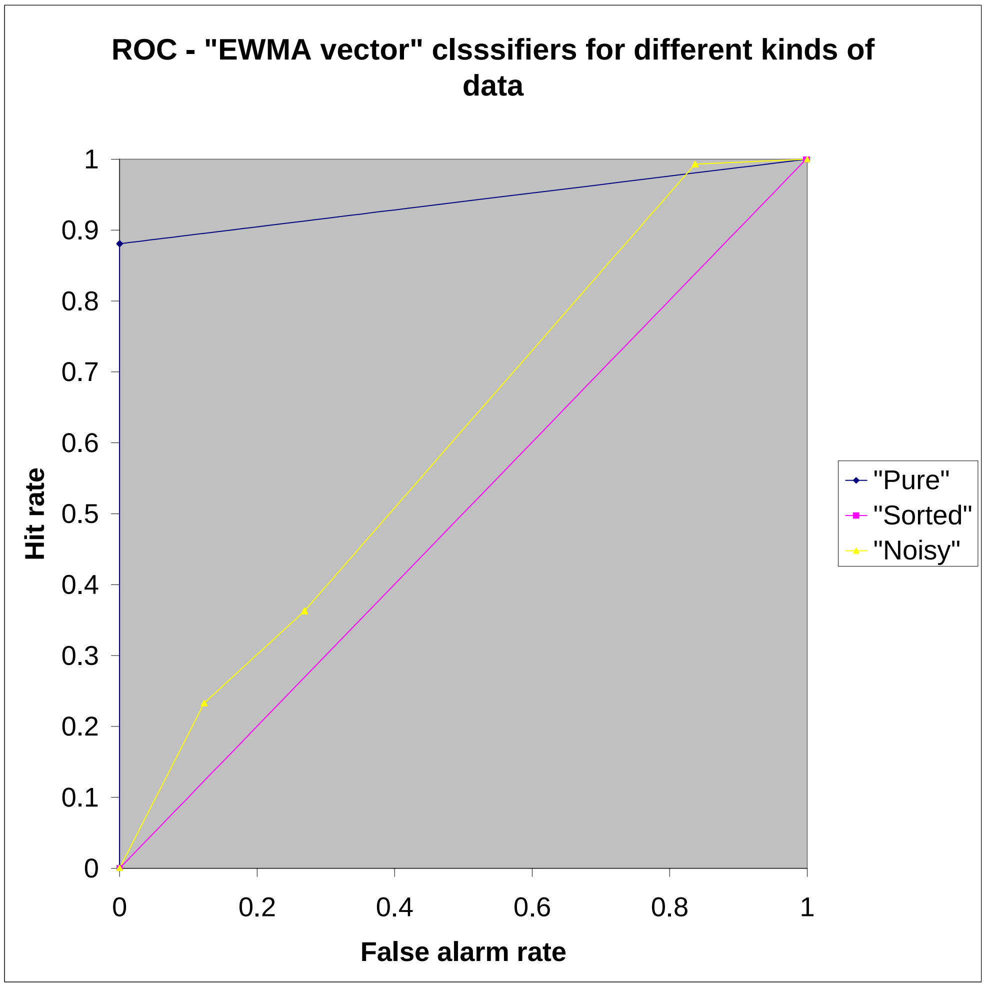


Figure 5-10. ROC curves of the "EWMA vector" classifiers based on the CHAID algorithm for different kinds of data.

Chapter 6: A System Architecture for Computer Intrusion Detection

6-1 Related Work

There are surveys on existing host-based, network-based and router-based IDSs [67, 4]. In this paper we focus on host-based IDSs. Host-based IDSs monitor activities occurring on individual host machines in an information system through computer audit data. A detailed description of computer audit data is provided in the next section.

Table 6-1. Characteristics of existing host-based IDSs.

IDS	**System Architecture**	**Intrusion Detection**	**Fusion**
CI	Centralized	Anomaly detection	
CSM	Distributed on target hosts	Signature recognition	
DPEM	Distributed on target hosts	Anomaly detection	
COAST-EIMDT	Distributed on target hosts		
EMERALD	Distributed on target hosts and security servers	Signature recognition Anomaly detection	
GrIDS	Centralized	Anomaly detection	
Haystack	Centralized	Signature recognition Anomaly detection	Fixed weights
Hyperview	Centralized	Signature recognition Anomaly detection	
IDES/NIDES	Centralized	Signature recognition Anomaly detection	
IDIOT	Centralized	Signature recognition	
NADIR	Centralized	Signature recognition	
RIPPER	Centralized	Signature recognition Anomaly detection	
USTAT	Centralized	Signature recognition	

Note: a blank cell in the table indicates that either information or technology is not available.

Table 6-1 summarizes the characteristics of some host-based IDSs. Those IDSs are distinguished with regard to the intrusion detection technique, the information fusion technique, and the distribution of detection functions.

Intrusion detection techniques generally fall into two categories: signature recognition and anomaly detection. Signature recognition (also called "misuse detection" in some literature) techniques identify and store signature patterns of known intrusions, match activities in an information system with known patterns of intrusion signatures, and signal intrusions when there is a match. Signature recognition techniques are efficient and accurate in detecting known intrusions, but cannot detect novel intrusions whose signature patterns are unknown. Anomaly detection techniques establish a profile of a subject's normal activities (a norm profile), compare observed activities of the subject with its norm profile, and signal intrusions when the subject's observed activities differ largely from its norm profile. The subject may be a user, file, privileged program, host machine, or network. Anomaly detection techniques can detect both novel and known intrusions if they demonstrate large deviations from the norm profile. Since anomaly detection techniques signal all anomalies as intrusions, false alarms are expected when anomalies are caused by behavioral irregularity instead of intrusions. Hence, signature recognition techniques and anomaly detection techniques are often used together to complement each other.

When both a signature recognition technique and an anomaly detection technique are used in an IDS to monitor the same observed activities in an information system, the two different techniques may produce different evaluation results. An information fusion technique is required to combine different results from different intrusion detection techniques on the same observed activities into a composite score for an overall assessment of intrusion likelihood with the observed activities. Very few IDSs incorporate an information fusion technique. The existing information fusion techniques for intrusion detection are based on mainly a fixed weight vector that assigns a fixed weight to the result from an individual intrusion detection. The fixed weight reflects the relative importance of an individual intrusion technique in producing the overall assessment of intrusion likelihood.

Existing IDSs also differ in their architecture to support various methods of distributing intrusion detection functions. Two common architectures are: centralized and distributed. A centralized architecture distributes intrusion detection functions usually on a security server. A distributed architecture usually distributes intrusion detection functions to security modules running on the target host machines in an information system. Sometimes, security servers are also used in a distributed architecture to coordinate security modules and perform further analysis.

Each IDS listed in Table 6-1 is briefly described below to give some concrete pictures of how IDSs work. The Purdue University COAST (Computer Operations, Audit, and Security Technology) Laboratory's Enhanced Intrusion and Misuse Detection Techniques program (COAST-EIMDT) [68] relies on multiple independent entities, called autonomous agent, working collectively. Autonomous agents perform certain security monitoring functions at each host. They are independently running entities, i.e., their execution is scheduled only by the operating system, and not by other processes. Agents may or may not need data produced by other agents to perform their work and may receive high-level control commands from other entities. The agents can be added and removed from a system without altering other components and without restarting the IDS and can be tested on their own before introducing them into a more complex environment. The results collected from the agents are organized into a hierarchical manner to ensure scalability of the system. Hence, COAST-EIMDT uses a distributed architecture with intrusion detection functions on target hosts. Although COAST-EIMDT does not specify intrusion detection techniques that it may use, any intrusion detection can fit into this architecture to carry out the intrusion detection functions.

IDES (Intrusion Detection Expert System) [78] is a real-time intrusion detection expert system, which has now been merged into NIDES [74], the Next generation IDES. The audit data collected in the hosts is securely transmitted to NIDES via the network and processed to provide real-time response. The statistical engine of IDES/NIDES verifies each new audit record against the norm profiles for both the subject and the group the subject belongs to. It also verifies each session against the norm profile of sessions when the session completes. IDES/NIDES also consists of an expert system that keeps signature patterns of known intrusions. Hence, IDES/NIDES uses a centralized architecture of intrusion detection functions, and employs both signature recognition and anomaly detection techniques. When different measures of the same observed activities are used to detect intrusions, IDES/NIDES proposes a squared sum of the results from individual measures to generate a composite score. IDES/NIDES does not address the combination of the results from its statistical engine and the expert system for signature recognition.

GrIDS (Graph based Intrusion Detection System) [69] employs data source modules that run on each host and report relevant information to graph engines that build a graph representation of activity in the network and compare the graph with the norm profile for detecting intrusions. Hence, GrIDS uses a centralized architecture, and employs mainly an anomaly detection technique for intrusion detection.

NADIR (Network Anomaly Detector and Intrusion Reporter) [72] performs distributed data collection by employing the existing service nodes in Los Alamos National Laboratory's

Integrated Computer Network (ICN) to collect audit information, which is then analyzed by a central expert system that compares the summary of the weekly audit data with rules in the expert system describing violations of security policies and suspicious activities. Hence, NADIR uses a centralized architecture, and employs a signature recognition technique for intrusion detection.

In the peer-based IDS described in [81] Cooperative Security Managers (CSM) are employed to perform distributed intrusion detection that does not need a hierarchical organization or a central coordinator. Each CSM performs as a local IDS for the host where it runs, but can additionally communicate with other CSMs and exchange information about users moving through the network and detect suspicious activities based on signature recognition. Hence, CSM uses a distributed architecture of intrusion detection functions on the target host machines, and employs a signature recognition technique for intrusion detection.

The IDS based on computer immunology (CI) emulates the biological immune systems to some extent by employing a set of “self” patterns of event sequences to detect “non-self” at various key locations in the information system [20]. CI uses a centralized architecture, and employs an anomaly detection technique for intrusion detection.

The EMERALD project [80] proposes a distributed architecture for intrusion detection that employs entities called service monitors that are deployed to hosts and perform monitoring functions. It also defines several layers of monitors for performing data reduction in a hierarchical fashion. Hence, EMERALD uses a distributed architecture of intrusion detection functions on the target host machines and some security servers, and employs both a signature recognition technique and an anomaly detection technique from IDES/NIDES. EMERALD does not address the fusion of results from different intrusion detection techniques for the same observed activities.

Hyperview [70] employs an artificial neural network component and an expert system component for intrusion detection. The expert system component monitors audit trails for known signatures of intrusion. The artificial neural network component adaptively learns the behavior of a user and raises alarms when the audit events deviate from the already learned behavior. Hyperview connects the artificial neural network to two expert systems. The first one monitors the operation and the training of the network and evaluates its output. The other scans the audit trail for known patterns of intrusions, and together with the output from the first expert system forms an opinion if it should raise an alarm. Hence, Hyperview employs both a signature recognition technique and an anomaly detection technique for intrusion detection, and uses a centralized architecture.

USTAT (Unix State Transition Analysis Tool) [73] is a prototype implementation of the state transition analysis approach to intrusion detection. The state transition analysis takes the view that the computer initially exists in a secure state, but as a result of a number of penetrations - modeled as state transitions - it ends up in a compromised target state. That is, state transitions are used to describe signature patterns of known intrusions. USTAT uses a centralized architecture, and employs only a signature recognition technique.

DPEM (Distributed Program Execution Monitoring) [21] focuses on the correct security behavior of the system, i.e., a security privileged application that runs on the system. DPEM reads the security specifications of acceptable behavior of privileged UNIX programs, and checks audit trails for security violations. The DPEM prototype monitors programs executed in a distributed system by collecting execution traces from the various hosts, and where relevant, distributing them across the network for processing. Hence, DPEM uses a distributed architecture, and employs an anomaly detection technique for intrusion detection.

IDIOT (Intrusion Detection In Our Time) [76] was developed at COAST (now the Center for Education and Research in Information Assurance and Security - CERIAS) in Purdue University. It employs Colored Petri-nets (CP-nets) for signature based (pattern matching) intrusion detection and suggests a layered approach for applying signature-based techniques. The patterns (signatures) are written in an ordinary textual language and then parsed, resulting in a new pattern-matching engine. This engine can then be dynamically added to an already running IDIOT instance via the user interface. The user can even extend IDIOT to recognize new audit events. Hence, IDIOT uses a centralized architecture, and employs a signature recognition technique for intrusion detection.

RIPPER [77] uses data mining for automatic and adaptive construction of intrusion detection models. It utilizes auditing programs for extracting extensive set of features to describe each network connection or host session, and applies data mining programs to learn rules that accurately capture the behavior of intrusions and normal activities. These rules can then be used for signature detection and anomaly detection. The RIPPER framework implements several algorithms. The user can pre-process audit data to provide the algorithms with a representation on a natural level. The user then gives RIPPER the task of automatically mining patterns of abusive or normal behavior. Hence, RIPPER uses a centralized architecture, and employs a data mining technique for both signature recognition and anomaly detection.

Haystack was developed as an audit data reduction and intrusion detection tool [82]. It collects audit data from a target host machine, and performs primitive statistical analysis to detect deviations of observed activities from normal activities and to compare observed activities with known intrusions. Haystack is the only IDS in Table 6-1 that incorporates an information fusion technique. Haystack calculates a weighted intrusion score by summing the results from multiple intrusion detection components. Fixed weights are assigned. Hence, Haystack uses a centralized architecture, and employs both a signature recognition technique and an anomaly detection technique for intrusion detection along with the fixed-weight technique for information fusion.

Table 6-1 does not include virus scanners and monitors. Most virus scanners and monitors are based on signature recognition techniques to capture signatures of viruses and use those signatures to detect viruses. Those virus scanners and monitors can also be considered as IDSs with focus on data in the file system and other media rather than computer audit data. Because virus scanners and monitors monitor data that are different from computer audit data monitored by host-based IDSs, virus scanners and monitors can be included in host-based IDSs as additional components to produce separate intrusion reports.

Many of the IDSs in Table 6-1 are still in a research prototype stage. Those commercial IDSs rely mainly on signature recognition techniques with the inherent limitation of detecting novel intrusions [4]. Some of the existing IDSs, such as Hyperview and EMERALD, incorporate both signature recognition techniques and anomaly detection techniques. However, if both signature recognition techniques and anomaly detection techniques are used together to monitor the same activities in an information system, we should expect different results from different intrusion detection techniques. The fusion of different results into a composite index of intrusion warning is necessary.

Hence, in addition to signature recognition techniques and anomaly detection techniques, our ISA-IDS also includes the function of information fusion. Moreover, the signature recognition techniques and the anomaly detection techniques in the ISA-IDS have significantly advanced the state-of-the-art of intrusion detection by overcoming problems with the existing intrusion detection techniques in performance and scalability [13, 14, 22, 83, 84, 85]. The main driver of developing the ISA-IDS lies in the advantage of these intrusion detection techniques. Currently, the ISA-IDS focuses on computer audit data, and uses a distributed architecture. ISA-IDS places little overhead on the target information system, and uses independent host machines (security servers) to perform intrusion detection. ISA-IDS will have the capability of processing network traffic data in the future. The following sections provide more details in the design, implementation and testing of ISA-IDS.

6-2 Data Source

ISA-IDS is a distributed host-based IDS that has been designed to collect audit event data from many hosts. In the prototype implementation of ISA-IDS we have simplified the data collection process by using the audit data from the Defense Advanced Research Projects Agency (DARPA). This has been done to keep our focus towards implementing the centralized IDS server and the individual technique servers. In the next stage of research we will develop the event data collectors.

The Information Systems Technology Group of MIT Lincoln Laboratory (LL), under the DARPA Information Technology Office and Air Force Research Laboratory (AFRL) sponsorship, collected network traffic data and computer audit data containing both normal and intrusive activities from a simulation network, and developed a guideline for evaluating existing intrusion detection systems in 1998 [2]. The simulation network architecture to collect the network sniffed traffic data and audit data is shown in Figure 6-1. Figure 6-1 shows the points where network traffic data from a Sun OS sniffer and computer audit data from the Solaris Basic Security Module (BSM) are collected. The ISA-IDS prototype uses only the Solaris Basic Security Module (BSM) audit data from this 1998 DARPA-LL data set.

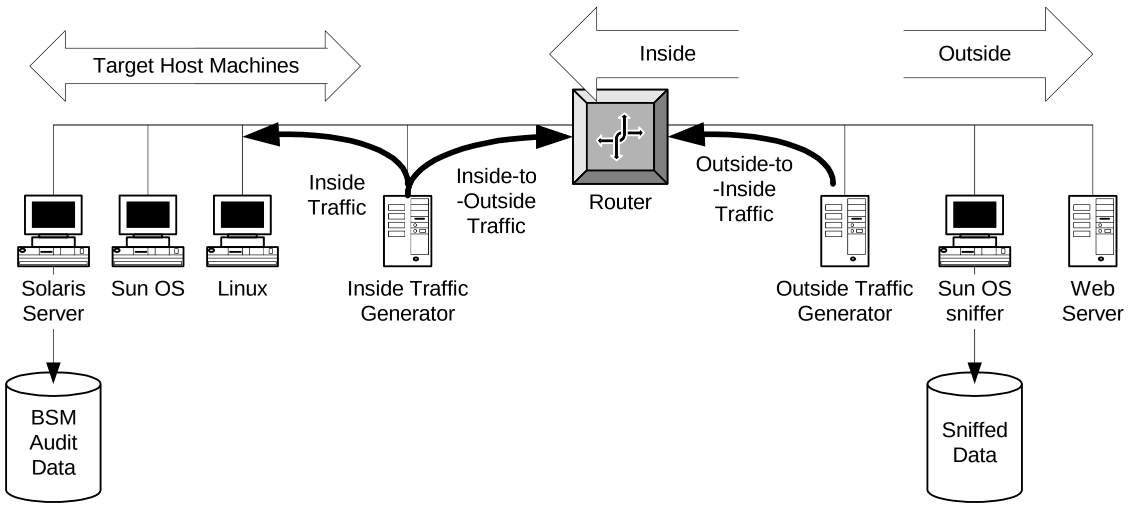


Figure 6-1. The simplified offline simulation network architecture for collecting network sniffed and audit data.

For this 1998 DARPA-LL data set, normal activities were simulated to resemble normal activities occurring in a real computer network at a typical Air Force base [2]. A large amount of web, telnet, and mail traffic was generated between the inside host machines and the outside host machines and web sites. Many user automata of various types such as secretaries, programmers and managers were also used to generate normal activities. Statistical profiles of various user types were used to simulate normal activities. These statistical profiles consider information such as the occurrence frequency of different UNIX commands, typical login times and session durations, and so on. Human actors were also used to perform more complex tasks such as upgrading software, adding users, remotely accessing programs, and performing other system administrative tasks. Those simulated normal activities have created the “white noises” background.

While simulating normal activities in the simulation network, intrusions have also been simulated on the “white noise” background. Simulated intrusions fall into four categories: denial of service attacks to disrupt a host or network service, remote-to-local attacks where an attacker without an account on a victim machine attempted to gain local access, user-to-root attacks where a local user attempted to gain privileges of the UNIX root user, and probe-scan attacks using programs to probe or scan a network for gathering information such as a topology of the network. More details about the simulation of normal and intrusive activities for the 1998 DARPA-LL data can be found in [75, 2, 79].

Normal and intrusive activities occurring on a Sun Solaris machine in the simulation network have been recorded by the Solaris auditing facility, BSM, producing the computer audit data with a mixture of normal and intrusive activities. BSM audit data consist of a sequence of audit files. Each audit file consists of many audit records. Each audit record captures an auditable event. Each audit record includes a sequence of audit tokens, each of which contains specific information about the attributes of the event. By an “event” we point to a particular activity in the underlying computer system at a certain time along with the information pertaining to that activity. “Attributes” of an event are the variables that contain information about such activity.

There are twenty-five different audit tokens. However, most audit records contain only a few tokens such as the header token with such information as an event ID indicating the type of audit event, the time when the record was created and so on, the subject token with such information as the user ID, the group ID, the session ID and so on, and a return token indicating the return status of the event.

The BSM of Solaris monitors each event on a host machine and generates a sequence of audit records. Auditable events are generated by system calls used by the kernel of the operating system or by application programs. BSM defines 284 different types of audit events. There are thousands of commands in Solaris UNIX. The audit events are more close to the kernel of the operating system. Hence, the type of event is more representative than the actual command sequences used. For example, we can use any text editor, such as, vi, ed, pico to edit a file, but most of the time the audit event stream will contain the following event types: AUE_EXECVE, AUE_OPEN_R, AUE_ACCESS, AUE_STAT.

The 1998 DARPA-LL data are provided in two parts – the training data set and the testing data set. The training data set is provided for developing an IDS system and configuring its parameters. The testing data set is provided for testing the performance of the trained IDS system. There are more than 300 instances of 38 different intrusions in the training and testing data. Using the provided description of intrusions in the training data from the MIT Lincoln Laboratory, we can label each audit record and its corresponding audit event in the training data set as either normal or intrusive along with the intrusion name. The training data set covers intrusive activities occurring in the midst of normal activities for seven weeks. Another two weeks of network sniffed traffic data and BSM audit data containing both normal and intrusive data are provided as the testing data. This data set contains all those intrusions that were in the training set and also new types of intrusions. The data records in the testing data set are not labeled. The IDS are allowed to scan though this data set only once, and the IDS has to label, using its algorithms, each data record as normal or intrusive.

The ISA-IDS prototype conforms to this specification from the MIT Lincoln Laboratory for evaluating intrusion detection systems. The Centralized IDS Server of the prototype ISA-IDS scans through the training data set to set the parameters of the techniques we have integrated in the system through the Individual Technique Servers. The Centralized IDS Server then takes the testing data set, passes each audit data record to the Individual Technique Servers, fuses their results, and labels each audit data record as either normal or intrusive.

Since no label is provided for audit events in the testing data set of the 1998 DARPA-LL data, we do not know the ground truth of these audit events and thus cannot evaluate the performance of the ISA-IDS prototype with the ground truth. Therefore, we cannot use the testing data set for testing the ISA-IDS. Instead, we drew our two-day training data and two-day testing data from the seven-week training data set of the 1998 DARPA-LL data. Two-day data are used for training and testing to reduce the amount of time required to complete the training and testing. In the seven weeks of training data, there are in total 35 days of data. We picked four days of data as a representative of these 35 days of data. We wanted to pick two days where there were not many intrusions and two days where there were many intrusions. According to this criterion, we chose week-1, Monday data as day-1 data, week-4, Tuesday data as day-2 data, week-4, Friday data as day-3 data and week-6, Thursday data as day-4 data. These days are nonconsecutive, come from different weeks and each are of different weekday. We changed the day marks of day-2 data, day-3 data, and day-4 data so that four days become consecutive days.

6-3 ISA-IDS Architecture

This section elaborates the ISA-IDS system architecture. Figure 6-2 sketches the system architecture. It consists of one Centralized IDS Server (CIS) that collects the audit events from the hosts that it is monitoring. We deploy an Event Data Collector (EDC) in each of the host machines. The Event Data Collector collects the BSM audit data from the host and sends audit events to the Central Event Collector (CEC) of the Centralized IDS server. The Central Event Collector in turn sends them through the Event Dispatcher to the Individual Technique Servers

(ITS) that use different intrusion detection algorithms. Each Individual Technique Server provides its response to the Centralized IDS Server by calculating the intrusion warning (IW) level for that event. After getting back the results, the Centralized IDS Server fuses the different IW levels from the Individual Technique Servers for the same event and produces a composite IW level for that event.

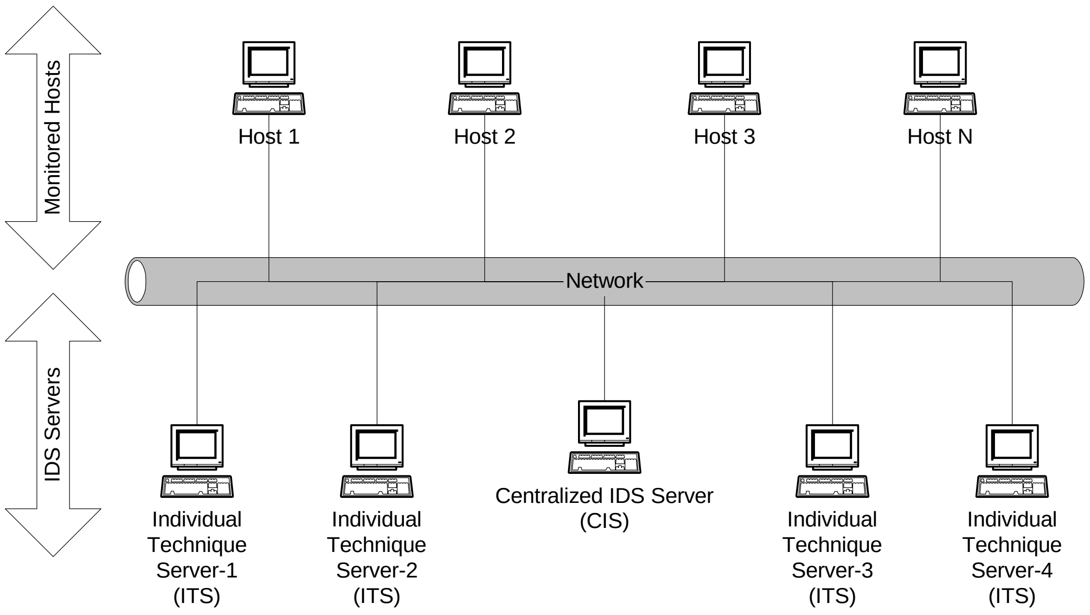

Monitored Hosts
Host 1
Host 2
Host 3
Host N
Network
IDS Servers
Individual Technique Server-1 (ITS)
Individual Technique Server-2 (ITS)
Centralized IDS Server (CIS)
Individual Technique Server-3 (ITS)
Individual Technique Server-4 (ITS)

Figure 6-2. System architecture of the ISA-IDS

.

The Centralized IDS Server acts like a centralized control center of the entire network system. Its main tasks are the followings in order (Figure 6-3 shows the data flow diagram of the ISA-IDS):

1. Collect audit event data from the hosts through the Central Event Collector;
2. After data pre-processing, dispatch the audit event stream to the Individual Technique Servers through the Event Dispatcher;
3. Collect the IW levels for events from the Individual Technique Servers;

4. Fuse the IW levels and produce a composite binary IW level (1 for intrusive and 0 for normal) for each event;
5. Label the input event as intrusive or normal according to the composite IW level and produce an alert signal if required.

Therefore, after deploying the Event Data Collectors in the host machines, setting up the Centralized IDS Server and the Individual Technique Servers and starting to run the system, the SSA only has to check the Centralized IDS Server from time to time for any possible intrusion alert.

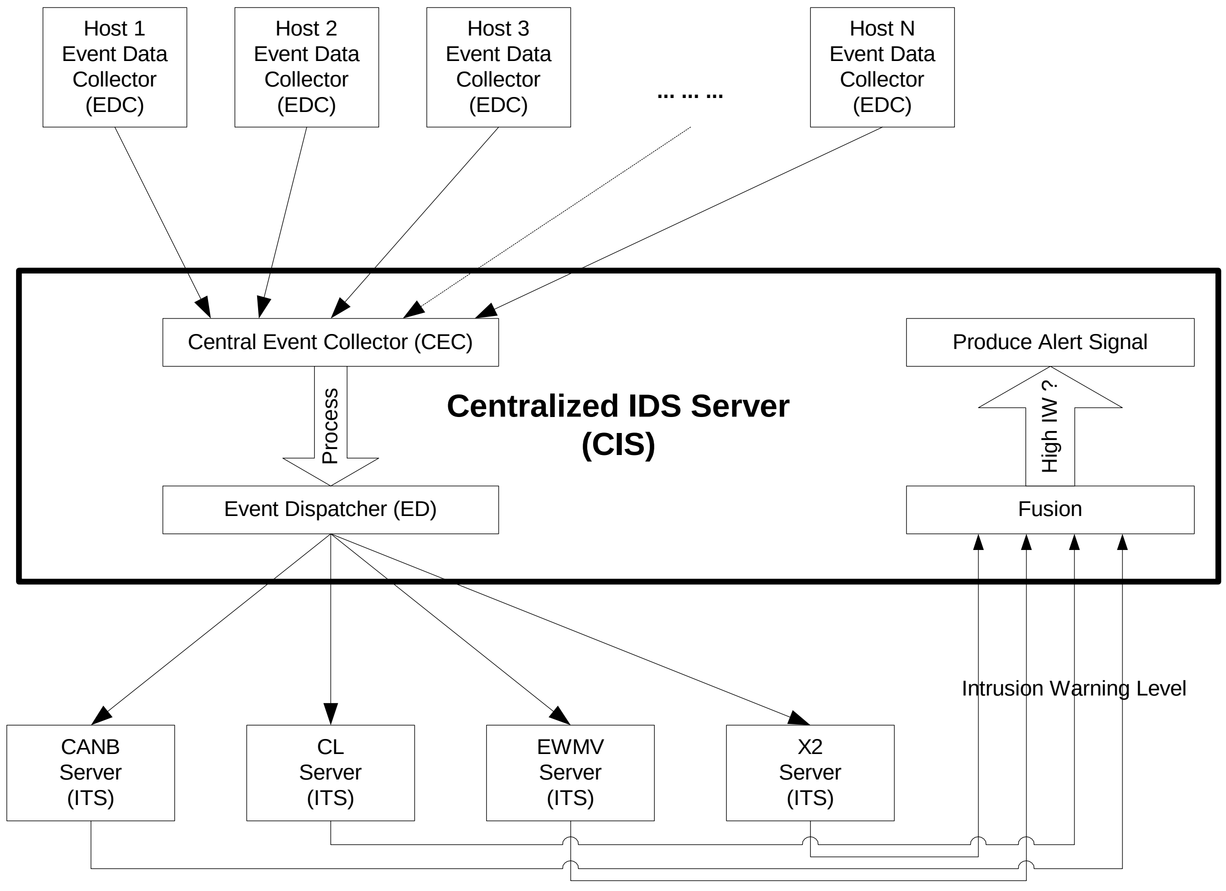

Host 1 Event Data Collector (EDC)
Host 2 Event Data Collector (EDC)
Host 3 Event Data Collector (EDC)
... ...
Host N Event Data Collector (EDC)
Central Event Collector (CEC)
Process
Event Dispatcher (ED)
Centralized IDS Server (CIS)
Produce Alert Signal
High IW ?
Fusion
Intrusion Warning Level
CANB Server (ITS)
CL Server (ITS)
EWMV Server (ITS)
X2 Server (ITS)

Figure 6-3. Data flow diagram of the ISA-IDS.

The SSA may have to manually inspect all the alarms raised by the ISA-IDS to figure out the cause of the intrusion. Sometimes the ISA-IDS may produce a warning signal for an event that is not in fact intrusive - this is an example of a false alarm. Sometimes the ISA-IDS may not produce a warning signal for an event that is in fact intrusive – this is an example of a miss. The lower the false alarm rate and the miss rate, the better the ISA-IDS.

At the beginning we need to train the Individual Technique Servers using the training data. Initially, a training data set containing normal and intrusive activities, such as the 1998 DARPA-LL data, can be collected and used for training. After the initial training the system can use the audit data collected from the hosts in the monitored information system. Since the Individual Technique Servers have already been trained, they can produce intrusion labels for each of the events in the event stream. During off-peak hours, say after midnight, the data collected during the day can be used for updating or retraining the system parameters of each of the techniques inside Individual Technique Servers. The retraining can be scheduled on a per day, per week or per month basis as desired by the system administrator. Currently the ISA-IDS prototype uses the training part of the 1998 DARPA-LL audit data set to train the Individual Technique Servers and labels the testing data set during testing.

For an Individual Technique Server supporting an anomaly detection technique, the training of the technique requires only computer audit data of normal activities. Because each audit record in the training data set of the 1998 DARPA-LL data can be labeled as normal or intrusive, we can filter the training data set by its label so that we have computer audit data of only normal activities to train this anomaly detection technique. For an Individual Technique Server supporting a signature recognition technique, the training of the technique requires computer audit data of both normal and intrusive activities. The training data set of the 1998

DARPA-LL data can be used directly without filtering to train the technique. Details of the Individual Technique Servers are provided in the following sections.

In an environment where there is an absence of data from sources where intrusions are known and characterized, two approaches can be taken to train the Individual Technique Servers for signature recognition techniques. In the first approach, the Individual Technique Servers for signature recognition techniques in the ISA-IDS can be disabled at first. Therefore, the ISA-IDS relies on the Individual Technique Servers to label incoming audit records. When intrusions are detected, the Individual Technique Servers for signature recognition techniques can use the labeled audit records to discover the signatures of those known intrusions and thus get trained. In the second approach, because many intrusion scenarios that happened in the past are known, those intrusion scenarios can be simulated just as what LL did to create computer audit data of intrusive activities. These two approaches can be used alone or together.

In the past few years we have investigated several techniques for intrusion detection covering both anomaly detection and signature recognition. The detailed description and performance results of the techniques can be found in [13, 14, 22, 83, 84, 85]. Four of these techniques have been incorporated into the ISA-IDS prototype. The techniques are named - CANB for Canberra metric, CL for Clustering algorithm, EWMV for Exponentially Weighted Moving Variance technique and X2 for Chi-square test. These techniques and their implementation are briefly described in the next section.

Since the amount of data to be processed is huge, and each of the techniques needs a considerable amount of disk space and processing power we have used a separate machine for each of the techniques. Therefore one technique corresponds to one Individual Technique Server. The Centralized IDS Server must have a very high processing power and very large amount of disk space to store all the incoming events and results produced. Separating the techniques into different Individual Technique Servers enables us to comfortably plug-in new kinds of intrusion detection techniques with minor modification in the event dispatcher and fusion technique of the Centralized IDS server. Note that if desired, the Centralized IDS Server and Individual Technique Servers can reside on the same machine rather than on different machines.

6-4 Prototype Implementation and Intrusion Detection Techniques

In the prototype implementation we have used the 1998 DARPA-LL BSM audit data and have not yet incorporated the host Event Data Collectors. Except the Event Data Collectors in the hosts and the Central Event Collector in the Centralized IDS Server we have implemented all other components. We have implemented the Centralized IDS Server and the Individual Technique Servers with Visual C++™ and for communication between them over the network we have used the Orbix™ implementation of CORBA. Following CORBA standards, the Individual Technique Servers have been implemented as CORBA servers and the Centralized IDS Server has been implemented as a CORBA client. The Centralized IDS Server scans through the BSM audit events and dispatches the event objects to the servers through the Event Dispatcher by calling appropriate server services that the Individual Technique Servers implement. All the Individual Technique Servers have been implemented in a uniform way so that new Individual Technique Servers can be easily plugged in.

We can model the entire network as a system and the hosts that we are monitoring as independent subsystems. Since the 1998 DARPA-LL MIT data contains BSM audit data only from a single host machine with the Solaris operating system, there is currently only one subsystem in the prototype. When there are audit data from a network of multiple host machines, the ISA-IDS can also set up the system and multiple subsystems.

We have analyzed the incoming BSM audit events from two perspectives - "Event Type" [13, 14] and "Process Model" [85]. Note that the SSA can select either of the two methods to use for an information system. Figure 6-4 highlights the difference between these two methods through the simplified class diagram of the prototype.

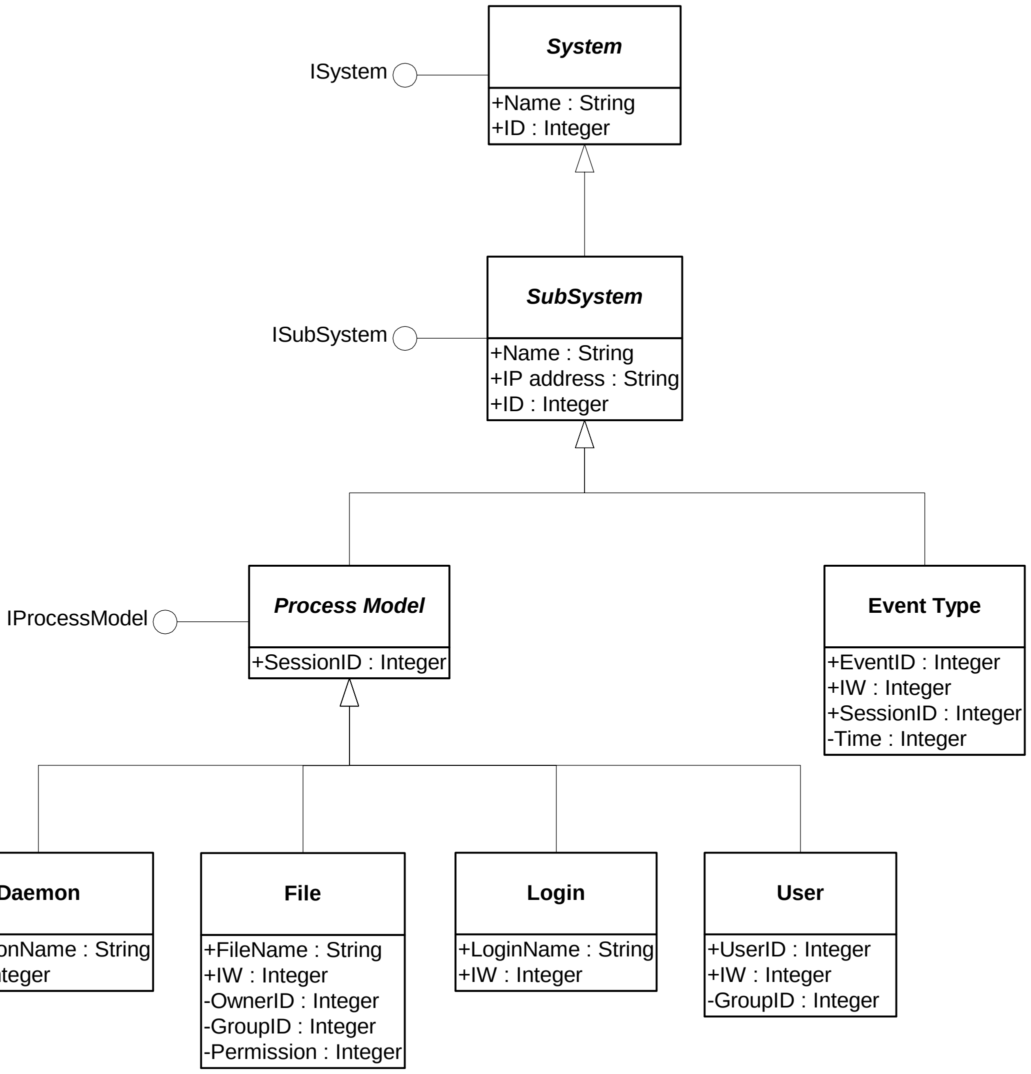


Figure 6-4. Simplified class diagram of the ISA-IDS.

In the Event Type method, we look at the type and time of event occurrence and pass that information to the Individual Technique Servers. We ignore other information related to that event. This way of analysis is much simpler and faster but ignores details of each event. For example, if the event is related to a file that a user is using, Event Type analysis does not care about which user is using which file. Its analysis is based entirely on the sequence of the type of events.

According to Process Model, we look more into the events and find out whether each event is related to Login or File or User or any of the Daemon objects. An event can involve more than one of these object classes. For example, if user A accesses file B - it corresponds to an event related to a user object (A) and also to a file object (B). Therefore, the Process Model method of data analysis not only considers the type of the event but also the actual source of the event and gives more information to the Individual Technique Servers. It is more complex than the Event Type, and therefore requires much more processing power.

In the Process Model method, depending on the objects (Daemon, File, Login, User) associated to an event, we create those objects involved in the event. In the Event Type method, we have only one type of object to create – the subsystem for the single host machine. After filling up their attributes properly, the Event Dispatcher of the Centralized IDS Server sends those objects to the Individual Technique Servers. Each Individual Technique Server scans its database looking for the previous occurrence of that object and retrieves its profile and sets the Intrusion Warning (IW) level ($0 \leq IW \leq 1$, higher value indicates more intrusiveness) according to the algorithm of that Individual Technique Server. The Individual Technique Server then returns the object back to the Centralized IDS Server with the IW level filled in. After collecting the IW levels from all the Individual Technique Servers, the Centralized IDS Server feeds these into the fusion component to produce a composite IW level. If the level is 1, a signal is generated notifying the SSA to take further action. Figure 6-5 illustrates the procedures taken in the Process Model method or the Event Type method to calculate the IW value.

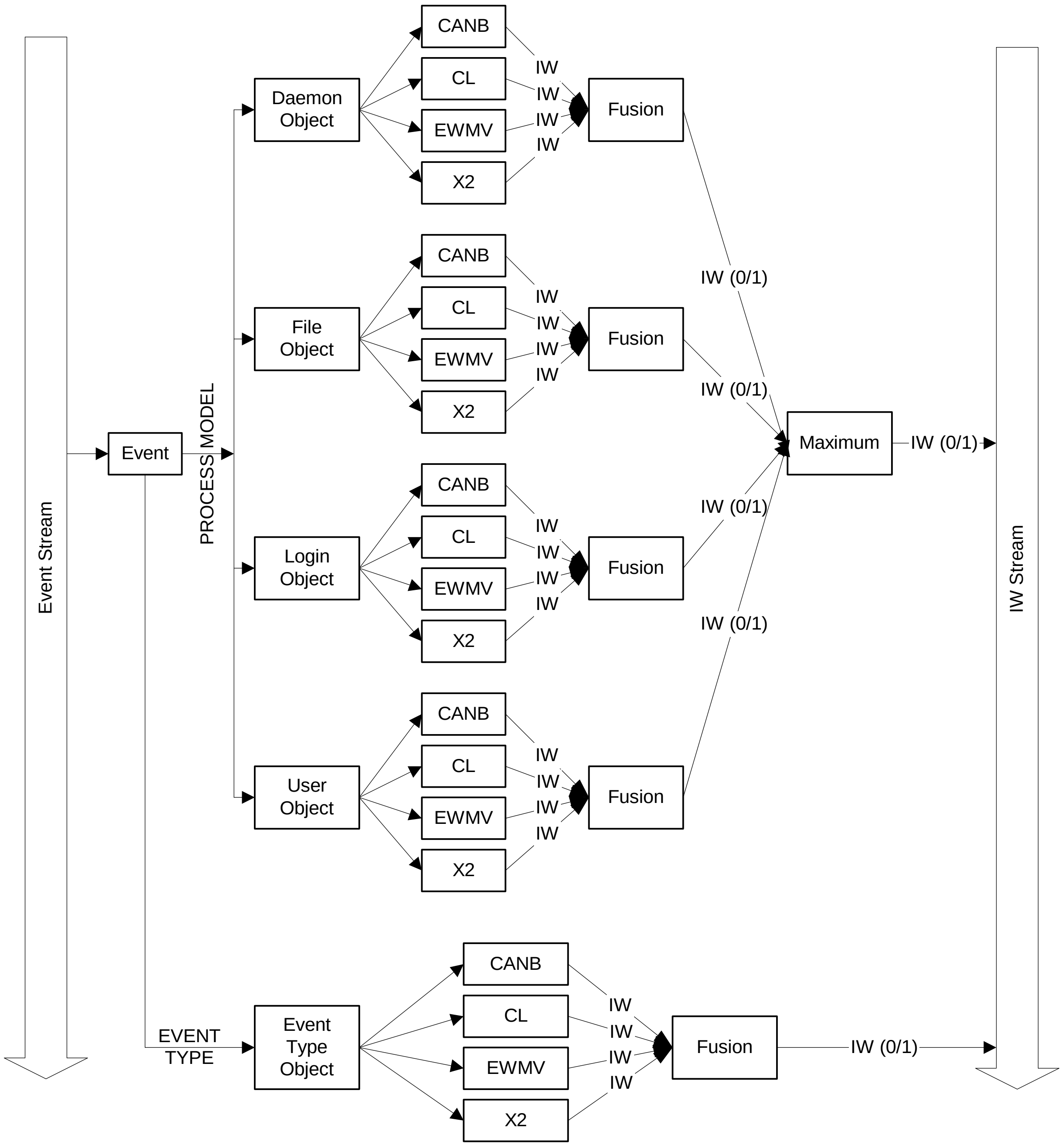


Figure 6-5. IW calculation in Process Model and Event Type methods.

The rest of the section describes the four Individual Technique Servers we have incorporated - CANB, X2, EWMV and CL.

Each object associated with an event has several attributes, which we refer as variables. For example, in the Event Type method, we have 284 event types and therefore there are 284 variables, numbered from 1 to 284. In the Process Model method, the number of variables depends on the type of the objects (Daemon, File, Login, or User) associated with the event. We have defined the variables for each of the objects after thoroughly studying the characteristics of each object class.

The training consists of several phases, and is slightly different from one Individual Technique Server to another, though they follow the same procedure. In phase I, the Event Dispatcher of the Centralized IDS Server takes the events chronologically one by one and creates associated objects and passes them to the Individual Technique Servers. For each event on each object, an Individual Technique Server calculates an observation value of the variables for the event on the object, updates the training parameters for the Individual Technique Server, and stores the observed value (we refer to it as *X* value) and the training parameters. In phase II, the Individual Technique Server takes the *X* value stream and computes the *T* value according to the equation of a specific technique along with the average and standard deviation of the T value stream, if needed (for CANB, X2 and CL). The average and standard deviation of the T value stream are used to determine the control limits of the *T* value for that technique.

In testing, the *X* value and the *T* value are computed for each audit event. The control limits of the *T* value from phase II of the training are then used to transform the T value into the IW value that indicates whether or not to signal an intrusion.

Now we briefly discuss the algorithm in each of the Individual Technique Servers during training and testing. During testing only X values, T values and IW levels are calculated. All the training parameters are calculated in training.

The CANB is an anomaly detection technique. It builds a norm profile in training using only audit data of normal activities. The norm profile is then used in testing to determine the IW level of each audit event. During training and testing, the CANB technique uses the Exponentially Weighted Moving Average (EWMA) method [24] to compute an observed value – the X value - for the event on an object. X is a vector consisting of multiple variables. The observed value of a variable indicates the occurrence frequency of activity that the variable represents. For example, in the Event Type method, the observed value of the 284 variables is computed for each audit event. The observed value of variable i represents the occurrence frequency of event type i in the recent past. In training, only audit data of normal activities are used to build the norm profile. In phase I of training, the X values for all the audit events in the training data set and the mean vector of the X values for those audit events are computed. Phase II goes over the X value stream from phase I, and transforms the X value for the n^{th} event into the Canberra distance metric from the mean vector of the X values to produce the T value for this event. Hence, the T value measures the distance of the X value for this event to the mean vector of X according to the Canberra distance definition. Meanwhile, the mean and standard deviation of the T values are updated incrementally by taking into account the T value for this event. After the last event in the training data set, the incrementally updated mean and standard deviation of the T values become the mean and standard deviation of the T values for all the training data, $\overline{T}$ and S_T. This mean and standard deviation of the T values are used to determine the control limits as follows: $\left[\overline{T} - CLF * S_T, \overline{T} + CLF * S_T\right]$ which indicates the range of the *T* values for normal activities, where CLF is a control limit factor, and is usually set to 2 or 3.

In testing, the *X* value and the *T* value are computed for each audit event. If the *T* value falls outside the range of the control limits (the range of normal activities), the IW level is set to 1 (a signal indicating an intrusion); otherwise, the IW level is less than 1 (no signal). The specific IW level depends on how close the T value is to the center of the range of the control limits (how close the *T* value is to the mean vector of *T*).

The X2 technique is an anomaly detection technique similar to the CANB technique. The two techniques differ only in the definition of the distance metric that is used to compute the *T* value for an event. For the X2 technique, the chi-square distance metric is used.

Both the CANB technique and the X2 technique monitor the frequency distribution of multiple variables (i.e., for various event types in the Event Type method) and detect anomalies as intrusive. SSA can select either of the two techniques when running the ISA-IDS. Because these two techniques are similar, there is no need to select both.

The EWMV technique is an anomaly detection technique. It uses the Exponentially Weighted Moving Variance technique to monitor the event intensity for an object. It is designed to detect significant changes in the event intensity. Many denial-of-service intrusions manifest through changes in the event intensity. The only information that the EWMV technique uses of each event is the time of event occurrence. Therefore the EWMV technique uses only one variable for the *X* value and does not calculate the *T* value. The EWMV technique uses a time decaying method to calculate the observation value reflecting the intensity of events in the most recent past (the frequency count of events per time unit). The exponentially weighted moving average (EWMA) and the exponentially weighted moving variance (EWMV) are then calculated to set the control limits as follows: [EWMA – CLF*EWMV, EWMA + CLF*EWMV].

In testing, the IW level is computed by comparing the observed event intensity with the control limits. If the *X* value for an event falls outside the control limits, the IW level is set to 1 (a signal for intrusion); otherwise, the IW level is less than 1 (no signal).

The CL technique is a signature recognition technique that learns intrusion signatures from the training data of both normal and intrusive activities, and uses the learned signatures to detect intrusions in testing. The CL technique employs a clustering and classification algorithm to cluster audit events in the training data set and to classify each audit event in the testing data set into normal or intrusive. The CL technique uses the same method of computing the observation value – the X value – as in the CANB and X2 techniques. The variables associated with an event are considered as dimensions. Hence, an observation is a data point in a multi-dimensional space. In training, the CL technique groups data points in the training data set into normal and intrusive clusters. In testing, the CL technique uses the cluster structure to determine the IW level of an event by examining whether the observation value for the event is close to normal clusters or intrusive clusters.

In the prototype of the ISA-IDS, we use either the X2 technique or the CANB technique to fuse the IW levels from the Individual Technique Servers into a composite IW level. When using the X2 technique or the CANB technique for information fusion, the IW levels from the Individual Technique Servers produce the *X* value consisting of multiple variables for those individual Technique Servers respectively. That is, the information fusion is based on the norm profile of the Individual Techniques Servers' outputs for normal activities in training. In testing, if the results from the Individual Technique Servers for an event deviate significantly from the norm profile, the information fusion technique produces a composite index triggering a signal for intrusion; otherwise, no signal is produced from the information fusion technique for the overall assessment of intrusion likelihood.

More details of the individual intrusion detection techniques and the information fusion technique can be found in [13, 14, 33, 83, 84, 85].

In testing, the Event Dispatcher of the Centralized IDS Server scans through the data only once. For each object associated with an event, Individual Technique Servers calculate observation values for the variables of the object. From the observation values an Individual Technique Server calculates the T value and then IW level ($0 \leq$ IW ≤ 1). The IW values calculated by the Individual Technique Servers are fed into fusion to get the composite IW level ($IW_{composite}$: 0 for normal, 1 for intrusive), which is used to label the testing data into normal or intrusive. The training process is complex and requires several passes over the input event stream. Training should be done periodically when requested or set by SSA.

Figure 6-6 presents the performance of the X2 technique based on the Receiver Operating Characteristic (ROC) analysis of session signal ratio. The CANB technique is similar to the X2 technique. Hence, it is not tested. Figure 6-7 presents the performance of the CL technique based on the Receiver Operating Characteristic (ROC) analysis of session signal ratio. The testing of information fusion is still ongoing, and thereby is not presented here.

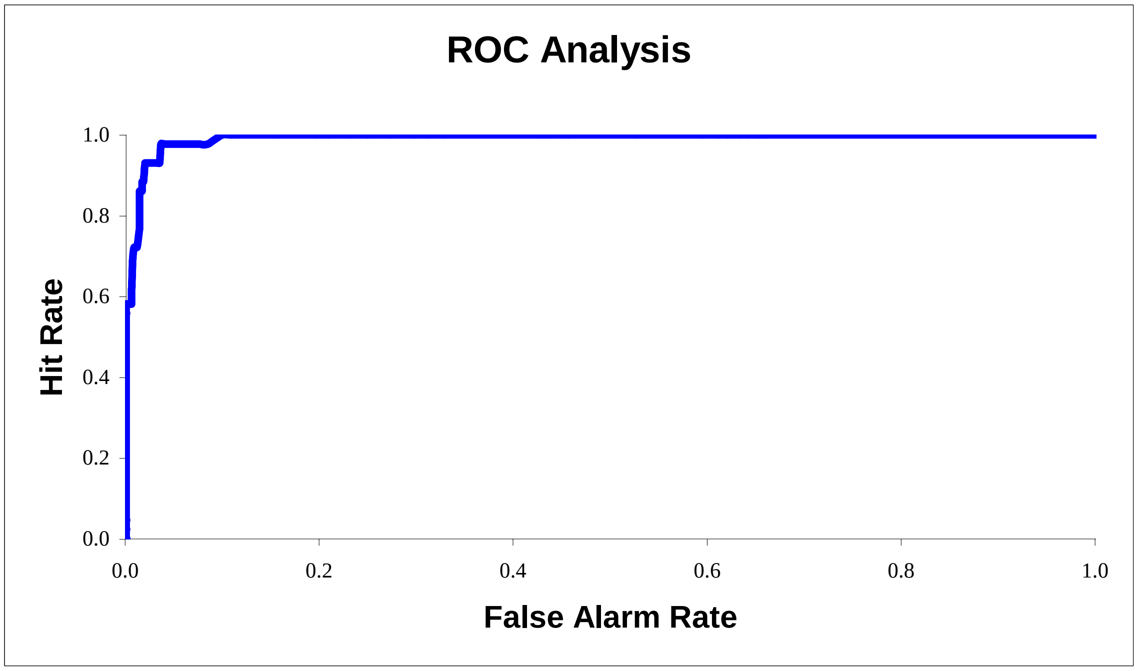


Figure 6-6. The performance of the X2 technique.

ROC analyzes the tradeoff between false alarm and hit rates for detection systems. A false alarm is a signal on an event when in fact the event is normal. A false alarm rate is ratio of the number of false alarms to the number of all the normal events in the testing data. A hit is a signal on an event when in fact the event is intrusive. A hit rate is the ratio of the number of hits to the number of all the intrusive events in the testing data. ROC was developed in the field of signal detection [46, 47]. It has now become the standard approach to evaluate detection systems. ROC curves for intrusion detection indicate how the hit rate changes as the signal threshold varies to generate more or fewer false alarms to tradeoff detection accuracy against analyst workload. Measuring the hit rate alone indicates only intrusions that an intrusion detection system may detect, and it does not indicate the human workload required to analyze false alarms generated by the normal background traffic. A low false alarm rate with a high hit rate means that the detection output can be trusted and human labor required to confirm any detection is minimized. For a given signal threshold, a pair of a hit rate and a false alarm rate can be computed. Hence, by varying the signal threshold, a curve (ROC curve) can be plotted. The closer the ROC curve is to the top-left corner representing the 100% hit rate and the 0% false alarm rate, the better the detection performance is.

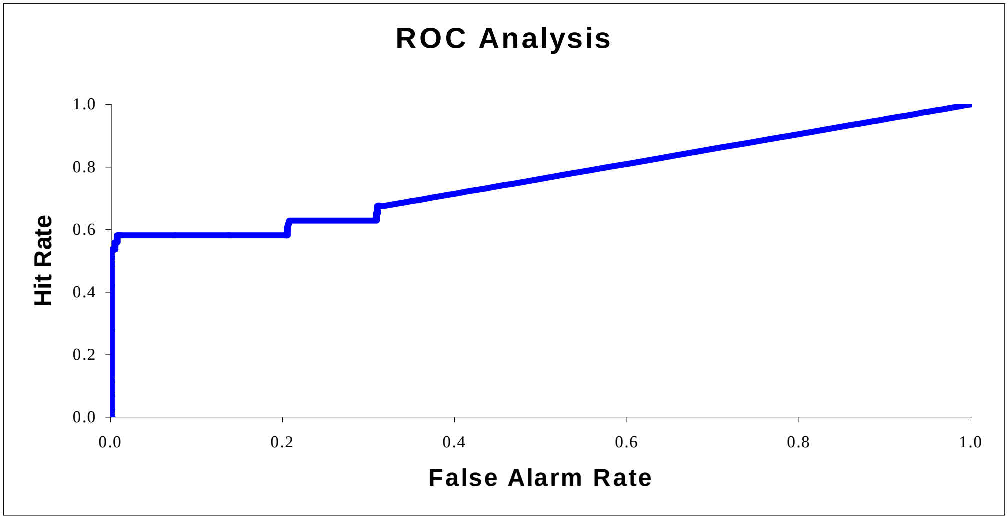


Figure 6-7. The performance of the CL technique.

Since intrusive events occur in sessions, we analyze the results in terms of session signal ratio. We group the events according to sessions and count how many of the events inside a session are signaled and we call the ratio of the number of signals to the number of events in a session as session signal ratio. If it is an intrusive session then we expect a high session signal ratio. If it is a normal session, we expect the session signal ratio to be low. In this paper we present the ROC curves for the X2 technique and the CL technique using the session signal ratio as shown in Figure 6-6 and Figure 6-7.

The X2 technique achieves the 60% hit rate at the 0% false alarm rate. At the 2% false alarm rate the hit rate of the X2 technique rises to 85%. The technique achieves the 100% hit rate after the 10% false alarm rate. Therefore, the X2 technique produces good detection performance.

When there is no false alarm produced by the CL technique, the hit rate reaches 53.5%. If we want to limit the false alarm rate to no more than 10%, we get the 58.1% hit rate.

Hence, the X2 anomaly detection technique has better intrusion detection performance than the CL signature recognition technique. This performance difference is expected because a signature recognition technique can detect only known intrusions used in the training, whereas an anomaly detection technique can detect both known and novel attacks if they demonstrate significant deviations from normal behavior.

The report in [2] presents the performance of some other IDSs evaluated. The performance evaluation of those IDSs is based on the whole set of the 1998 DARPA-LL training and testing data, whereas our training and testing of ISA-IDS are based on only four-day data. Hence, we cannot draw conclusions with regard to the comparison of those IDSs and ISA-IDS in their intrusion detection performance. If we simply examine the ROC curves of those IDSs and ISA-IDS, the X2 anomaly detection technique performs much better than the anomaly detection techniques in those IDSs in [2]. The CL signature recognition does not perform as well as the best signature recognition technique in those IDSs in [2]. However, all the signature recognition techniques in those IDSs in [2] uses some form of manual tuning (e.g., the manual input of rules, and the manual selection of features) to achieve better performance, whereas the training of our CL signature recognition technique was done automatically without any manual tuning.

6-5 Summary

In this paper we have presented the architecture and implementation details of ISA-IDS that is tested using the 1998 DARPA-LL audit data. During training it calculates the training parameters for each of the techniques. During testing, for each audit event on each object ISA-IDS applies a selective set of four intrusion detection techniques to find out the level of intrusion and fuses them into a single IW value, 0 for normal and 1 for intrusive. In ISA-IDS, we incorporate both a signature recognition technique and several anomaly detection techniques along with an information fusion technique. The techniques in ISA-IDS are scalable to large amounts of audit data, where many existing signature recognition techniques and anomaly detection techniques cannot scale to large audit data in real time.

The ISA-IDS prototype currently uses the 1998 DARPA-LL BSM audit data for testing. The data come from one single host machine. We can incorporate data from multiple host machines as well. The Event Data Collector has to be developed and installed in each of the hosts. The system currently uses off-line audit data. The testing can be run in real time, and the training can be run periodically.

References

1. Stallings W. *Network and Inter-network Security Principles and Practice.* Prentice Hall: Englewood Cliffs, NJ, 1995.
2. Lippmann R, Fried D, Graf I, Haines J, Kendall K, McClung D, Weber D, Webster S, Wyschogrod D, Cunningham R, Zissman M. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*. IEEE Computer Society: Los Alamitos, CA: IEEE Computer Society, pp. 12-26, January 2000.
3. Kaufman C, Perlman R, Speciner M. *Network Security: Private Communication in a Public World.* Prentice Hall: Englewood Cliffs, New Jersey, 1995.
4. Debar H, Dacier M, Wespi A. Towards a taxonomy of intrusion-detection systems. *Computer Networks* 1999 31: 805-822.
5. Escamilla T. *Intrusion Detection: Network Security beyond the Firewall.* John Wiley & Sons: New York, 1998.
6. Vigna G, Eckmann S, Kemmerer R. The STAT Tool Suite. In *Proceedings of the DARPA Information Survivability Conference and Exposition.* IEEE Computer Society: Los Alamitos, CA, pp. 46-55, January 2000.
7. Kumar S. *Classification and Detection of Computer Intrusions.* Ph.D. Dissertation, Department of Computer Science, Purdue University, West lafayette, Indiana, 1995.
8. Lee W, Stolfo SJ, Mok K. Mining in a data-flow environment: Experience in network intrusion detection. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99),* San Diego, CA, August, 1999, http://www.cs.columbia.edu/~sal/JAM/PROJECT/.

9. Anderson D, Frivold T, Valdes A. *Next-generation Intrusion Detection Expert System (NIDES): A Summary.* Technical Report SRI-CSL-97-07. Menlo Park, CA: SRI International, May 1995.

10. Neumann P, Porras P. Experience with EMERALD to date. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, April 1999, pp. 73-80, http://www.csl.sri.com/neumann/det99.html/.

11. Ye N, Li X, Chen Q, Emran SM, Xu M. Probabilistic techniques for intrusion detection based on computer audit data. *IEEE Transactions on Systems, Man, and Cybernetics* 2001 **31**(4).

12. Ye N, Chen Q. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International* 2001 **17**(2): 105-112.

13. Ye N, Chen Q, Emran SM, Noh K. Chi-square statistical profiling for anomaly detection. In *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. West Point: United States Military Academy, 2000.

14. Ye N, Chen Q, Emran SM, Vilbert S. Hotelling's T^2 multivariate profiling for anomaly detection. In *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. West Point: United States Military Academy, 2000.

15. Denning DE. An intrusion-detection model. *IEEE Transactions on Software Engineering* 1987 **SE-13**(2): 222-232.

16. Ghosh AK, Schwatzbard A, Shatz M. Learning program behavior profiles for intrusion detection. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and*

Network Monitoring, Santa Clara, California, April 1999, http://www.rstcorp.com/~anup/.

17. Javitz HS, Valdes A. The SRI statistical anomaly detector. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy.* May 1991.
18. Javitz HS, Valdes A. *The NIDES Statistical Component Description of Justification.* Technical Report A010. Menlo Park, CA: SRI International, March 1994.
19. Jou Y, Gong F, Sargor C, Wu X, Wu S, Chang H, Wang F. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. In *Proceedings of the DARPA Information Survivability Conference and Exposition.* Los Alamitos, CA: IEEE Computer Society, pp. 69-83, January 2000.
20. Forrest S, Hofmeyr SA, Somayaji A. Computer immunology. *Communications of the ACM* 1997 40(10): 88-96.
21. Ko C, Fink G, Levitt K. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy,* pp. 134-144, 1997.
22. N. Ye, X. Li, and S. M. Emran. “Decision trees for signature recognition and state classification,” In Proceedings of the IEEE SMC Information Assurance and Security Workshop, West Point, New York, June 2000.
23. Ye N, Chen Q. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International* 2001 **17**(2): 105-112.
24. T. P. Ryan, *Statistical Methods for Quality Improvement,* 1989; John Wiley & Sons.

25. D. C. Montgomery, *Introduction to Statistical Quality Control,* 1997; John Wiley & Sons.

26. J. S. Hunter, "The exponentially weighted moving average", *Journal of Quality Technology,* vol 18, 1986, pp 203-209.

27. S. W. Roberts, "Control chart tests based on geometric moving averages", *Technometrics,* vol 1, 1959, pp 239-251.

28. D. C. Montgomery, C. M. Mastrangelo, "Some statistical process control methods for autocorrelated data", *Journal of Quality Technology,* vol 23, num 3, 1991 Jul, pp 179-193.

29. C. M. Borror, D. C. Montgomery, G. C. Runger, "Robustness of the EWMA control charts to Non-normality", *Journal of Quality Technology,* vol 31, num 3, 1999, pp. 309-316.

30. S. H. Steiner, "EWMA control charts with time-varying control limits and fast initial response", *Journal of Quality Technology,* vol 31, num 1, 1999, pp. 75-86.

31. J. F. MacGregor, T. J. Harris, "The exponentially weighted moving variance", *Journal of Quality Technology,* vol 25, num 1, 1993, pp. 106-118.

32. S. S. Prabhu, G. C. Runger, "Designing a multivariate EWMA control chart", *Journal of Quality Technology,* vol 29, num 1, 1997 Jan, pp 8-15.

33. D. C. Montgomery, L. A. Johnson, J. S. Gardiner, *Forecasting and Time Series Analysis,* 1990; McGraw-Hill.

34. W. L. Winston, *Operations Research: Applications and Algorithms.* Belmont, CA: Duxbury Press, 1994.

35. P. Buttorp. Stochastic Modeling of Scientific Data. London: Chapman & Hall, 1995.

36. Gelman, Carlin, Stern, and Rubin. Bayesian Data Analysis. Chapman & Hall, 1995.

37. I. L. MacDonald, and W. Zucchini. Hidden Markov and Other Models for Discrete Valued Time Series. London: Chapman & Hall, 1997.

38. G. Vigna, and R. Kemmerer. "NetStat: A network-based intrusion detection appoach." In Proceedings of the 14th Annual Computer Security Applications Conference, Scottsdale, Arizona, December 1998, http://www.cs.ucsb.edu/~kemm/netstat.html/.

39. T. M. Mitchell, *Machine Learning.* Boston, MA: McGraw-Hill, 1997.

40. F. V. Jensen, An Introduction to Bayesian Networks. London: UCL Press, 1996.

41. B. H. Kantowitz and R. D. Sorkin. *Human Factors: Understanding People-System Relationships.* New York: John Wiley & Sons, 1983.

42. Hotelling H. Multivariate quality control. In Eisenhart C, Hastay MW, and Wallis WA, eds. *Techniques of Statistical Analysis.* McGraw-Hill: New York, NY, 1947.

43. Daniel WW. Biostatistics: A Foundation for Analysis in the Health Sciences. John Wiley & Sons: New York, NY, 1987.

44. Johnson RA, Wichern DW. *Applied multivariate Statistical Analysis*. Prentice Hall: New Jersey, 1998.

45. Ryan TP. *Statistical Methods for Quality Improvement*. John Wiley & Sons: New York, 1989.

46. Swets JA. The Relative Operating Characteristic in Psychology. *Science* 1973 **182**: 990–1000.

47. Egan JP. *Signal detection theory and ROC-analysis*. Academic Press: New York, 1975.

48. H. Debar, M. Dacier, M. Nassehi, and A. Wespi. "Fixed vs. variable-length patterns for detecting suspicious process behavior," In Proceedings of the 5th European Symposium

on research in Computer Security, Louvain-la-Neuve, Belgium, September 16-18, 1998, pp. 1-15.

49. C. Warrender, S. Forrest, and B. Pearlmutter. “Detecting intrusions using system calls: Alternative data models,” In Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 133-145.

50. F. V. Jensen. *The Instruction to Bayesian Networks*. New York: Springer, 1996.

51. J. Suzuki. “Learning Bayesian belief networks based on the MDL principle: An efficient algorithm using the branch and bound technique.” *IEICE Transactions on Information and Systems,* Vol. E82-D, No. 2, pp. 356-367, February, 1999.

52. J. Liu, and M. C. Desmarais. “A method of learning implication network from empirical data: Algorithm and Monte-Carlo simulation-based validation*.” IEEE Transactions on Knowledge and Data Engineering,* 9(6), November/December, pp. 990-1004, 1997.

53. W. L. Buntine. “Operations for learning with graphical models*.” J. of Artificial intelligence Research,* 2, pp. 159-225, 1994. Http://www.cs.washington.edu/research/jair/home.html.

54. T. M. Mitchell. *Machine learning.* Boston: McGraw-Hill, 1997.

55. J. Cheng, D. Bell, and W. Liu. “Learning Bayesian networks from data: An efficient approach based on information theory.” Http://www.cs.ualberta.ca/~jcheng/lab.htm.

56. R. Hofmann, and V. Tresp. “Discovering structure in continuous variables using Bayesian networks”. In *Advances in Neural Information Processing System* 8, Cambridge MA: MIT Press, 1996. Http://www7.informatik.tu-muenchen.de/~hofmannr.

57. U. M. Fayyad, and K. B. Irani. “Multi-interval Discretization of continous-valued attributions for classification learning.” In R. Bajcsy (Ed.), *Proceedings of the 13th*

International Joint Conference on Artificial Intelligence. Morgan-Kaufmann, pp. 1022-1027.

58. U. M. Fayyad, and K. B. Irani. "On the handling of continuous-Valued attributes in decision tree generation" *Machine learning,* 8(1), pp. 87-102, January, 1992.
59. Breiman, L., Friedman, J., Olshen, R. and Stone, C. *Classification and Regression Trees*, Wadsworth Inc., 1984.
60. Safavian, S. R., and Landgrebe, D., A Survey of Decision Tree Classifier Methodology, *IEEE Transactions on Systems, Man and Cybernetics*, No. 21, pp. 660-674, 1991.
61. S. L. Crawford. Extensions to the CART algorithm. *Int. J. Man-Machine Studies*, 31, pp. 197-217, 1989.
62. SPSS Inc. *User's Guide of AnswerTree 2.0*, Chicago, Illinois, 1998.
63. Ye, N. and Li, X. Application of Decision Tree Classifier to Intrusion Detection, In *Proceedings of Second International Conference on DATA MINING 2000*, Cambridge University, UK, July 2000.
64. Ye, N. and Li, X. Decision Tree for Signature Recognition and State Classification, *Information Assurance and Security Workshop of IEEE Systems, Man, and Cybernetics*, West Point, New York, USA, June 2000.
65. Brockwell, P. J., and Davis, R. A., Time Series: Theory and Methods, Springer-Verlag, 1987.
66. T. P. Ryan. Statistical Methods for Quality Improvement. New York: John Wiley & Sons.

67. Axelsson, S. (1998). Research in Intrusion-Detection systems: A Survey. *Technical Report 98-17*. Dept. of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

68. Balasubramaniyan, J., Garcia-Fernandez, J.O., Isacoff, D., Spafford, E.H., & Zamboni, D. (1998). An Architecture for Intrusion Detection using Autonomous Agents. *Technical Report 98-05*. Center for Education and Research in Information Assurance and Security, Department of Computer Sciences, Purdue University.

69. Chen, S.S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., & Zerkle, D. (1996). GrIDS: A graph based intrusion detection system for large networks. *Proceedings of the 19th National Information Systems Security Conference*, volume 1, pp. 361-370. National Institute of Standards and Technology.

70. Debar, H., Becker, M., & Siboni, D. (1992). A neural network component for an intrusion detection system. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 240-250.

71. Durst, R., Champion, T., Witten, B., Miller, E., & Spagnuolo, L. (1999). Testing and evaluating computer intrusion detection systems. *Communications of the ACM*, *42*(7), pp. 53-61.

72. Hochberg, J., Jackson, K., Stallings, C., McClary, J.F., DuBois, D., & Ford, J. (1993). NADIR: An automated system for detecting network intrusion and misuse. *Computers and Security*, *12*(3), pp. 235-248.

73. Ilgun, K. (1993). USTAT: A real-time intrusion detection system for UNIX. *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pp. 16-28.

74. Jagannathan, R., Lunt, T., Anderson, D., Dodd, C., Gilham, F., Jalali, C., Javitz, H., Neumann, P., Tamaru, A., & Valdes, A. (1998). System Design Document: Next-Generation Intrusion Detection Expert System (NIDES). *Technical Report*. Computer Science Laboratory, SRI International, 1998.

75. Kendall, K. (1999). A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. *M.S. Thesis*. MIT Department of Electrical Engineering and Computer Science.

76. Kumar, S., & Spafford, E.H. (1995). A software architecture to support misuse intrusion detection. *Technical report*. The COAST Project, Department of Computer Sciences, Purdue University.

77. Lee, W., Stolfo, S.J., & Mok, K. (1999). A Data Mining Framework for Building Intrusion Detection Models. *Proceedings of IEEE Symposium on Security and Privacy.*

78. Lunt, T.F., Tamaru, A., Gilham F., Jagannathan, R., Jalali, C., & Neuman, P.G. (1992). A real-time intrusion-detection expert system (IDES). *Technical Report Project 6784*. Computer Science Laboratory, SRI International.

79. Northcutt, S. (1999). *Network Intrusion Detection: An Analyst's Handbook*. New Riders Publishing.

80. Porras, P.A., & Neumann, P.G. (1997). EMERALD: Event monitoring enabling responses to anomalous live disturbances. *Proceedings of the 20th National Information Systems Security Conference*. National Institute of Standards and Technology.

81. White, G.B., Fisch, E.A., & Pooch, U.W. (1996). Cooperating security managers: A peer-based Intrusion Detection System. *IEEE Network*, pp. 20-23.

82. Smaha, S. (1988). Haystack: An intrusion detection system. *Proceedings of the IEEE Forrth Aerospace Computer Security Applications Conference.*

83. Ye, N. (2000). A Markov Chain Model of Temporal Behavior for Anomaly Detection. *Proceedings of IEEE Systems, Man and Cybernetics Information Assurance & Security Workshop*. United States Military Academy.

84. Ye, N., & Li, X. (2000). Application of Decision Tree Classifier to Intrusion Detection. *Proceedings of Second International Conference on DATA MINING 2000*. Cambridge University.

85. Ye, N., & Xu, M. (2000). A Process Model Based Integration of Intrusion Detection Techniques. *Proceedings of Industrial Engineering Research Conference.*